UiO : Department of Informatics
University of Oslo

The Nao Robot as a Platform for Evolutionary Robotics

(⌐°.°)

Snorre Alm Harestad

Master's Thesis Spring 2015

The Nao Robot as a Platform for Evolutionary Robotics

Snorre Alm Harestad

4th May 2015

Abstract

The Nao robot seemingly has capabilities of doing interesting Evolutionary Robotics research and experiments. Implementing an evolutionary learning platform capable of such research has therefore been realised in this project. After this, finding suitable optimisation problems for the Nao was done, before conducting evolutionary experiments such as locomotion gait optimisation involving both a physical and simulated version of the robot. The results from a preliminary experiment showed a clear difference in performance between the real robot and the one in the simulator. Appropriate methods and techniques were therefore investigated and conducted, resulting in more or less satisfying results in the end. The overall impression from the experiments is that the Nao robot is highly capable of doing Evolutionary Robotics research, and the implemented evolutionary learning platform should be a great start for conducting further and more interesting experiments in the future. Suggestions on further experiments and improvements can be found in the last chapter, along with a more detailed discussion of the already done experiments and results.

Acknowledgement

I would like to thank my supervisor, associate professor Kyrre Glette for suggestions and help in choosing interesting topics to explore and test out. I have especially appreciated your support and guidance throughout the work on the thesis.

I also would like to thank my fellow students for great discussions as well as professional and moral support, in addition to the awesome espresso machine located outside the lab. I would not have managed without you.

Contents

1 Introduction **1**
- 1.1 Purpose of the project . . . 1
- 1.2 Goals of the project . . . 2
- 1.3 Outline . . . 3

2 Background **5**
- 2.1 Evolutionary Algorithms . . . 5
  - 2.1.1 Overview . . . 5
  - 2.1.2 Genetic Algorithms . . . 6
  - 2.1.3 Evolutionary Multi-objective Optimisation . . . 8
  - 2.1.4 Non-dominated Sorting Genetic Algorithm II . . . 9
- 2.2 Evolutionary Robotics . . . 10
  - 2.2.1 The Reality gap problem . . . 11
  - 2.2.2 The Transferability Approach . . . 14
- 2.3 Humanoid robots and the Nao platform . . . 17
  - 2.3.1 Humanoid robots . . . 17
  - 2.3.2 The Nao robot . . . 18
- 2.4 Bipedal locomotion . . . 22
  - 2.4.1 Traditional approaches to walking methods . . . 22
  - 2.4.2 ZMP- and Inverse Pendulum-based approaches for stability . . . 23
  - 2.4.3 Other approaches . . . 24
  - 2.4.4 The Nao walking method . . . 25
- 2.5 Evolutionary Robotics on humanoids and the Nao . . . 32
  - 2.5.1 Evolutionary CPG with increased double support time (2011) . . . 32
  - 2.5.2 Automated CPG-Based walking for the Nao (2012) . . . 33
  - 2.5.3 Evaluation of Walk Techniques for the Nao (2011) . . . 34
  - 2.5.4 The standings today and the future for the Nao and other humanoid robots . . . 35

3 Implementation **37**
- 3.1 Capabilities of current available software packages . . . 37
  - 3.1.1 Simulation . . . 38
  - 3.1.2 Interfacing and integration of a robot control system 38
- 3.2 Implementation of an integrated evolutionary learning platform . . . 39

3.2.1 Choosing a capable learning module for parameter tuning . . . 40
3.2.2 A setup capable of evaluating solutions in both simulator and reality . . . 40
3.2.3 A suitable control system for the experiments . . . 41
3.2.4 A suitable simulator . . . 42
3.2.5 A suitable programming language . . . 43
3.3 Evolutionary Algorithms for the experiments . . . 45
3.3.1 NSGA-II . . . 45
3.3.2 Genetic operators . . . 45
3.3.3 Elitism . . . 49
3.4 A learning module with the capability of testing various EAs 49
3.5 Bipedal locomotion control using the NAOqi Framework . . 50

4 Experiments **55**
4.1 Purpose of the experiments . . . 55
4.2 Suitable optimisation problems for the Nao . . . 56
4.2.1 Bipedal locomotion learning . . . 56
4.2.2 Transferability Approach to deal with the Reality Gap 57
4.2.3 Locomotion learning on uneven terrain for stabilisation 59
4.2.4 Factors possibly affecting performance . . . 60
4.3 Integrated data collection . . . 61
4.3.1 During the learning loop in simulator . . . 61
4.3.2 During the physical robot testing on laminated floor 61
4.3.3 During testing in the Motion Capture lab . . . 62
4.4 Evolutionary optimisation experimental setups . . . 66
4.4.1 Preliminary investigation of walking gait optimisation using a simple GA . . . 66
4.4.2 Walking gait optimisation using NSGA-II . . . 66
4.4.3 Walking gait optimisation using the Transferability Approach . . . 67
4.4.4 Walking gait optimisation on uneven terrain . . . 67

5 Results and analysis **69**
5.1 Preliminary investigation - Using a simple GA with elitism in simulator . . . 70
5.1.1 Analysis . . . 71
5.2 NSGA-II results in simulator . . . 73
5.2.1 Analysis . . . 73
5.3 Gaits using evolved parameters from NSGA-II GA on laminated floor . . . 75
5.3.1 Analysis . . . 75
5.4 Gaits using evolved parameters from NSGA-II on carpet floor 76
5.4.1 Analysis . . . 77
5.5 Making a transferability function . . . 78
5.6 Transferability Approach results in simulator . . . 79
5.6.1 Analysis . . . 79
5.7 Transferability Approach gaits on laminated floor . . . 82

5.8 Transferability Approach gaits on carpet floor 82
5.8.1 Analysis 83
5.9 Gaits evolved on uneven terrain in simulator 84
5.9.1 Analysis 84
5.10 Gaits evolved on uneven terrain in MoCap lab 89
5.10.1 Analysis 89

6 Discussion **93**
6.1 General discussion 93
6.1.1 Preliminary investigation 93
6.1.2 Transferability Approach 94
6.1.3 Optimisation on uneven terrain 94
6.2 Conclusion 94
6.3 Future work 95

List of Figures

2.1 Genetic Algorithm flowchart 7
2.2 Two general procedures for system identification 13
2.3 System identification task for Evolutionary Robotics 14
2.4 Transferability function mapping 16
2.5 Multi-objective optimisation of fitness and transferability . . 17
2.6 The Nao humanoid robot. 20
2.7 The Nao robot's joints and motors 21
2.8 The DARwIn-OP open-source humanoid robot 23
2.9 Pattern generator ZMP tracking control 27
2.10 ZMP Preview Controller solved using Quadratic programming 29
2.11 Footstep placement using the NAOqi Footstep Planner . . . 31

3.1 UML object diagram of the evolutionary learning and testing. 41
3.2 Evolution of Nao walking gaits in the Webots simulator . . . 44
3.3 Single-point crossover GA operator 47
3.4 Uniform mutation GA operator 47
3.5 UML class diagram of the control system's use of external APIs 51
3.6 UML class diagram of an evolutionary learning module. . . 52

4.1 Uneven terrain used for gait optimisation in simulator . . . 60
4.2 The Nao in the laminated floor setup 62
4.3 OptiTrack FLEX:V100 high-speed Motion Capture camera . 64
4.4 The placement of passive markers on the Nao robot 64
4.5 Example of 3-D movement of the Nao using evolved gait parameters captured in the MoCap lab. 65

5.1 Evolution of fitness values using a simple GA in simulator . 71
5.2 Evolution of fitness values using NSGA-II in simulator . . . 74
5.3 The difference quotient between fitness values in simulation and reality 77
5.4 Plot showing gait performance between simulator, laminated floor and carpet floor 79
5.5 Interpolated transferability function using IDW 80
5.6 Transferability approach results 81
5.7 Gait performance differences between simulation and reality using Transferability approach. 83
5.8 Gait comparisons between two different approaches 85
5.9 The best solutions found using the Transferability Approach 86

5.10 Evolution of fitness values using the NSGA-II on uneven terrain in simulator 87
5.11 The Nao humanoid's axes and rotations 89
5.12 Gait optimisation results on uneven terrain tested on carpet floor. 90

List of Tables

2.1 A table with info about some known humanoid robots. . . . 19

3.1 A table showing all the gait parameters that can be altered or optimised using evolutionary optimisation plus the three velocity parameters in the top. 54

4.1 Experimental setup for the preliminary investigation using a simple GA for gait optimisation. 66
4.2 Experimental setup for NSGA-II with one objective 67
4.3 Experimental setup for the Transferability Approach 67
4.4 Experimental setup for the NSGA-II on uneven terrain . . . 68

5.1 Overview of all experiments conducted in this project. . . . 70
5.2 The best solution obtained using the NSGA-II with one objective in simulator in the top row, together with the default parameters. 73
5.3 Average movements for gaits evolved using the NSGA-II on laminated floor 76
5.4 Average movements for gaits evolved using the NSGA-II on carpet floor 78
5.5 Best solutions obtained using the NSGA-II on even and uneven terrain in simulator, together with the default parameters. 87
5.6 Solution 16 and 32 found in Figure 5.12 together with the default parameters. 91

Chapter 1

Introduction

The design of an "artificial brain" for a robot takes place in different fields of research such as Neuroscience[16] and Artificial Intelligence (AI)[61]. The most important one in this context is however in the field called Evolutionary Robotics (ER)[70][5][29]. This young field is about the use of Evolutionary Algorithms (EAs) in robotics, and can be said to be located in the engineering and science ecosystems. Related research fields are Machine Learning[49] on one side, and Developmental Robotics[10] on the other. A lot of research have been conducted in this field, although most of it is more proofs of concepts rather than solid knowledge. A major problem in this field is that most of the research is problem-specific, and as a consequence it makes it hard to re-use it for others.

1.1 Purpose of the project

The main purpose of this thesis is to be an initial investigation on how suitable the Nao robot and its platform are for applying Evolutionary Robotics and Learning Methods. It is therefore of interest not only to investigate the raw performance one can achieve on the platform by applying evolutionary learning techniques, but also to see how well the robot can deal with common challenges in Evolutionary Robotics. Issues that can raise are many, but the most interesting ones worth mentioning when it comes to evolution on humanoid robots are:

1. Robustness of the robot when using Evolutionary Algorithms.
2. Robot getting stuck or falling over during the evaluation period.
3. Wear and tear of the physical robot.
4. Accuracy of the simulation model, i.e. exploring the so-called Reality Gap.

Since the Nao previously has been utilised with success in experiments using Evolutionary Algorithms to e.g. make it walk in a stable way, e.g.

in [75], [47], [46] and [45], it is assumed that the robot and its platform should be able to handle these kind of experiments in a robust way. In this context, robustness can also be about the platform e.g. giving deterministic results, as well as dealing with the issues mentioned above. A newly published paper [64] seems promising for optimising fast walking gaits using novel methods, but was published too recently for it to be of any impact in this project. The Nao comes embedded with a lot of different sensors, and among them a sensor that can detect not only if the robot has fallen, but also if it is going to fall. It can also sense if it is stuck, by noticing that it is going to collide with a wall. By utilising these kind of sensors, it seems like one can minimise the impact challenge number two will have on the experiments. The Nao is an expensive robot, although the price is low compared to many other humanoids (see Table 2.1 for more details). Bearing this in mind, one therefore wants to minimise the number of experiments on the physical robot. The reason for this is to decrease the risk of getting parts, motors and actuators worn out, making the robot unstable or broken. Most of the experiments in this project are therefore carried out in a simulator. Exploring problems regarding this have as a consequence been investigated as well.

1.2 Goals of the project

To achieve what has been mentioned in the previous section, the main goal of this project is to find a good setup for Evolutionary Learning. Parts of this will be to investigate and choose from currently available software like robot controllers and simulators. When doing this, one will discover their suitability for Evolutionary Learning experiments, and also what one needs to implement oneself. Since one wants to find interesting experiments to carry out, e.g. what to learn, this investigation will naturally consist of some Evolutionary Robotics experiments as well.

To reach this goal of finding a good Evolutionary Learning setup, there are several tasks that need to be done. This includes:

- Making a learning module with the possibility of testing out various algorithms.
- Finding out what to learn, i.e. to find suitable optimisation problems for the Nao robot.
- Making it possible to evaluate and test solutions both on the real robot and in a simulator.
- Doing integrated data collection from the experiments, both in the learning loop, i.e. in simulation as well as for reporting afterwards. This will include Motion Capture data from testing on the real robot in the research group's own Motion Capture lab.
- Implementation of the system with an emphasis on concepts like

good programming interfaces, modularity, usability (ease of use) and documentation for future users.

1.3 Outline

This project consists of six different chapters: Introduction, background, implementation, experiments, results and analysis as well as conclusion.

Chapter 2 gives an introduction to the theory of the field called Evolutionary Robotics, along with its characteristics, advantages and problems. It also describes problems and suggested methods to solve them. It is also meant to cover humanoid robots including the Nao, in addition to give an overview over previously done and current research on the topics.

Chapter 3 covers the implementation of an integrated evolutionary learning platform for the experiments, with its features and advantages of using it.

Chapter 4 describes the experiments conducted during the project, along with their purposes and setups.

Chapter 5 lists up the results from these experiments, including an analysis of the outcomes.

Chapter 6 contains a more general discussion about the results and the project as well as a conclusion and a summary of the results and achievements made during the project. The last part of this chapter finishes the thesis by listing future interesting experiments and work that has not been conducted in this thesis. However, some of these would have been done if there was more time left.

Chapter 2

Background

This chapter is an investigation into the field of Evolutionary Robotics (ER), especially for humanoid robots and the Nao platform in particular[47]. It is also meant to give an overview of previous research on the subject, as well as the standings today. As you can see in some of the described research papers, they often consist of small experiments that demonstrate further research potential for different techniques, rather than solid knowledge. As most of the research in these papers are problem-specific and therefore of no re-use for others, the field is maturing slowly.

In this section you will also see descriptions of different methods that often are used for optimising the gait of humanoid robots like the Nao. They can be divided into four different strategies; mimicking of biological humans and animals, evolutionary optimisation of parameters and structures, the adoption of sound mechanical design rules as well as the optimisation of power-based indexes. This section concentrates on the first two of these, where examples are experiments using a Central Pattern Generator (CPG) and also Genetic Algorithms (GAs), respectively.

2.1 Evolutionary Algorithms

2.1.1 Overview

This section will be about Evolutionary Algorithms (EAs) in general, explaining what kind of algorithms they are, and advantages and disadvantages of such algorithms. It will focus on the variants that are most relevant for use in Evolutionary Robotics, such as Genetic Algorithms. Evolutionary algorithms, also known as evolutionary computation are basically algorithms inspired by Darwinian evolutionary systems, aiming to solve optimization problems with one or several criteria. From this it is reasonable to assume that an (artificial) evolutionary system should be made up of at least one population with individuals that compete for a restricted number of resources. These populations, due to birth and death of the individuals,

are dynamically changing. The ones that do not die, are either surviving into the next generation, or are used in reproduction. The new generation of offspring is therefore selected from either one or both these groups, and have often undergone modifications such as recombination and crossover. To choose which ones, we need to have a concept of fitness. This is like in Darwin's concept of natural selection[22], where only the fittest individuals survive and are used for reproduction. These reproduced individuals are more similar to their parents than when choosing randomly, and leads to the concept of variational inheritance; the offspring/children are not identical to their parents, but have similar qualities. One will therefore, at least in theory, get solutions that continually improve throughout the generations. However, there are some practical issues when using these types of algorithms[19]: It is up to the programmer to find good representations of the problem as well as the solutions; and what makes a solution better than another one? There are also challenges regarding the quality of the final solution, its robustness and also its generality to larger problems. Nevertheless, even though one is not guaranteed an optimal solution; it is generally more efficient than using iterative algorithms, and whenever one stops it from running, one will always get a more or less valid solution[19].

2.1.2 Genetic Algorithms

Genetic algorithms (GAs) is one of the most popular groups of techniques in the field of evolutionary computation. In general, it works on a randomly generated population consisting of artificial chromosomes or genotypes, which can be represented in different ways. Binary encoding is maybe the simplest one, where it for example can represent a variable in a function that needs to be optimised. It is also possible to encode several variables, such as the connection weights of a neural network. Then it selects and reproduces the chromosomes with the higher fitness[53], while also applying different genetic operators like crossover and mutation, which are described later. Fitness is a measurement of the performance of each individual artificial chromosome, where a higher value is better. This representation of a solution to the optimisation problem is usually called a phenotype. The process is continued until you have a wanted individual, i.e. a solution, or until the highest fitness value so far stops to increase. It is also problem-dependent; for example if you want to minimize the error between a function return and some target value, the fitness value will increase as the error decreases[53]. A flowchart of this process can be seen in Figure 2.1.

As good as Genetic Algorithms may seem, there are also people that raise questions about using it for artificial evolution. In [53], the authors argue that when the focus is shifted from system optimisation to self-organisation of autonomous systems like robots, the things that matter the most are different. When optimising systems, improving for example computational efficiency, GAs are used with success, and one can understand what

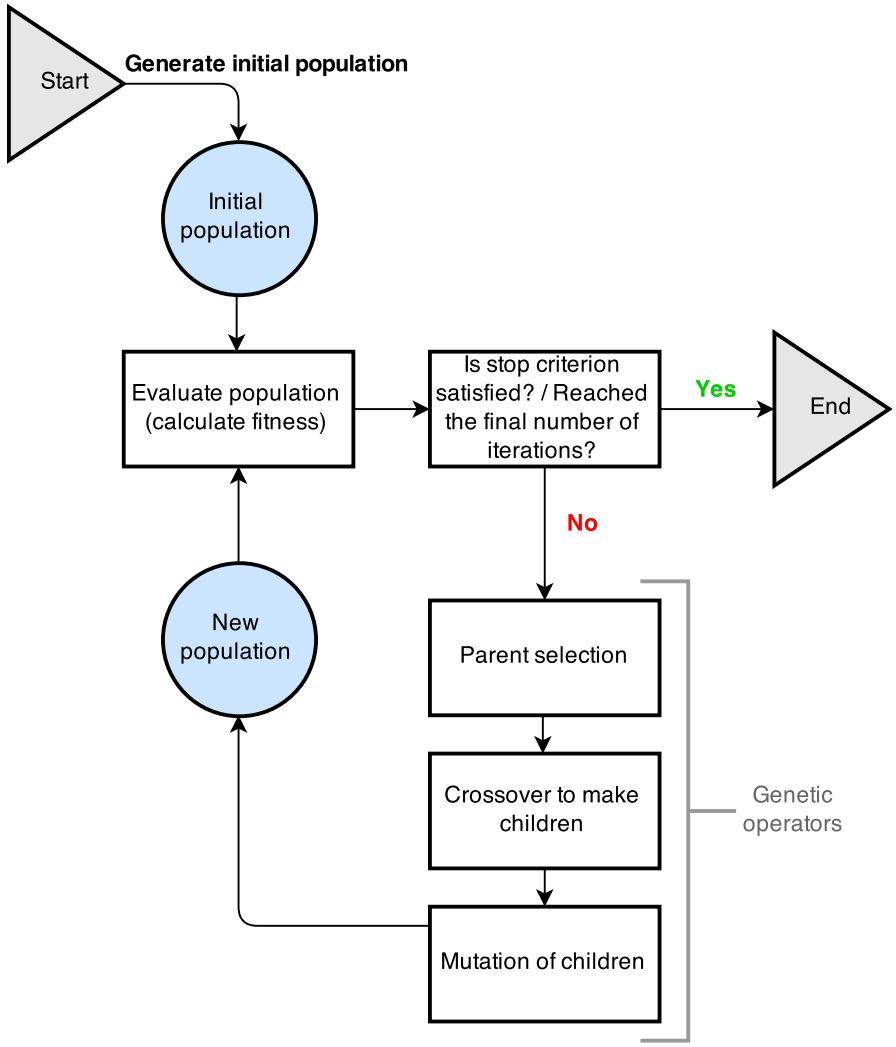


Figure 2.1: A flowchart showing a general GA approach. The stop criterion can be either a wanted fitness value, or until it stops increasing.

is going on. However when e.g. an autonomous robot interacts with its surrounding environment, the abilities to survive in unpredictable or unknown terrain and to find solutions to upcoming problems become more important. It is therefore often difficult to theoretically know what kind of skills one needs to optimise for, and the fitness function can be hard to constrain enough. A search using a hill climber in a restricted search space when considering these things suddenly seems not so useful any more. For that reason it is important for example to combine Genetic Algorithms together with other techniques, such as feedback from sensors monitoring the environment. An example of this is described later in this paper.

2.1.3 Evolutionary Multi-objective Optimisation

Evolutionary Multi-objective Optimisation (EMO)[11] is about optimising or solving problems with several objective functions using evolutionary techniques. Using EMO techniques to solve these kind of problems has in recent years become more and more popular among researchers in different fields, such as in engineering, the industry and scientific fields. This is like it has been for evolutionary single-objective optimisation, i.e. using ordinary Evolutionary Algorithms, for a long time. The optimisation problems' objective functions can have constraints, and they are often in conflict with each other. This means that there can not be one single solution to the problem; instead one ends up with a set of solutions that represent compromises between the different objectives. An optimal solution in this set is usually said to be Pareto optimal, which means that there is not another other solution where an increase in one criterion would not cause a decrease in at least one other criterion at the same time. In this Pareto optimal set, none of the solution points can clearly be said to be better than another, and their representing vectors are therefore said to be non-dominated, meaning they are not dominated by any other solution. If you plot these solution vectors, you get the Pareto front of the problem. To measure the performance of a Multi-objective Evolutionary Algorithm (MOEA), one is mainly interested in two things: Firstly, one wants to find as many elements of the Pareto optimal set as possible. Secondly, one is interested in maximising the spread of these found solutions, to make the distribution of vectors as smooth and uniform as possible. If you know the location of the global Pareto front, one is also interested in minimising the distance between it and the front found by the algorithm.

David Schaffers VEGA[63] from 1985 is considered to be the first real MOEA, and there have been developed several different ones after that. The first generation developed from 1989 to 1998 had a focus on simplicity, and notable ones are the Non-dominated Sorting Genetic Algorithm (NSGA), Niched- Pareto Genetic Algorithm (NPGA) and the Multi-objective Genetic Algorithm (MOGA). The second generation shifted the focus onto efficiency, and the most representative ones for this generation are probably the Strength Pareto Evolutionary Algorithm (SPEA), SPEA2, the Pareto Archived Evolution Strategy (PAES), the Multi-objective Evolutionary Algorithm based on Decomposition (MOEA/D) as well as the well-known NSGA-II, described in the next section. This one has been shown to perform well on such problems, nowadays the research is more focused on improving other aspects of the algorithms, such as many-objective problems, i.e. problems with four or more objectives. The makers behind the recently proposed NSGA-III for example claims to deal with this challenge.

An advantage of using EAs for these problems in comparison to e.g. mathematical programming techniques is that they can deal with a set of possible solutions, i.e. a population simultaneously. This makes one able

to find more than one element of the Pareto optimal set in one single run, often making them fast. Another benefit is their ability to deal with Pareto fronts that are concave or discontinuous.

2.1.4 Non-dominated Sorting Genetic Algorithm II

NSGA-II[42][4] stands for Non-dominated Sorting Genetic Algorithm version 2, and is one of the most popular MOEAs used in Evolutionary Robotics today. According to Google Scholar, it is quoted more than 13 thousand times in different papers and articles. It is based on the very effective first version called NSGA[69], but comes with several improvements. While the first version lacked elitism and was criticised for being slow, due to its complex non-dominating sorting algorithm, NSGA-II includes elitism and is said to have a better sorting algorithm. Elitism or elitist selection ensures that good solutions survive into new generations in an unaltered way, ensuring that the quality of the best found solutions in one generation are not becoming worse in the next one. The new version also removes the need of specifying a so-called sharing parameter. In 2014, a new version based on the NSGA-II framework was proposed[18]. While the NSGA-II gives satisfactory results on two and three objectives, this new version called NSGA-III has been developed in order to cope with many-objective optimisation problems, and is therefore only needed when having this many objectives.

How the NSGA-II works

If one runs the NSGA-II with only one objective, its behaviour is in practice like any other simple Genetic Algorithm that incorporates elitism: One first needs to specify the number of decision variables, e.g. walking gait parameters, as well as their corresponding ranges, if any. Then one needs to decide the number of individuals per generation and the total number of generations to evolve for. The first population is initialised with random parameters within their ranges, before it is sorted based on non-domination, i.e. for one objective only their fitness values. Since the optimisation uses only this one objective, only the best individual is placed in front one, unless there is another individual with the exact same fitness value. This is unlikely to happen in e.g. gait optimisation, since the fitness values are calculated from observed measurements. The second best individual is placed in front two, and so on. To maintain diversity among the different parameters, a crowding distance is calculated for each individual in each front, but because the fronts in most occasions only consist of one individual, this does not have any significant impact for single-objective optimisation. The offspring population is generated using crossover and mutation on the parent population, and the combined population is then sorted based on non-domination. The new generation is

made up of the N best individuals, where N is the population size, and this process continues until it reaches the final generation.

When using the NSGA-II with two or three objectives on the other hand, the behaviour differs from a simple GA in several ways. When the population is initialised and sorted based on non-domination, the first front contains the best solutions so far that do not dominate each other. These solutions get a fitness value of 1, while the second best solutions, i.e. the ones in front two are assigned fitness values of 2 and so on. The crowding distance between the individuals in each front is then calculated, and is a measure of how close an individual is to its neighbours, i.e. how similar they are. Since their fitness values are the same, larger crowding distances will make the population more diverse, making the risk of ending up with a Pareto-optimal front far from the superior one smaller. Parent selection using binary tournament is based on rank; the smaller number is better, and if the same rank, the greater crowding distance is better. Crossover and mutation works the same way as when having one objective, and parents and offspring are sorted based on non-domination again. The N best individuals are chosen from this combined population, and are used as the next generation.

2.2 Evolutionary Robotics

Evolutionary robotics (ER) deals with evolutionary techniques such as Evolutionary and Genetic Algorithms to develop autonomous robot control systems. It can roughly be divided into four different uses: The first one is parameter-tuning, which can be done either in a simulator, on a real robot or both. This approach is the most popular and developed one, and is described in more detail later in the chapter. Another use is something called evolutionary aided design, where the optimised parameters are not the goal, unlike in the previous mentioned approach. The parameters are here used for analysis to get better understandings of the underlying problem. This makes it easier to propose new solutions to the problem, which again can be optimised using an EA. The third way of using it is in embodied evolution or online evolution, where the algorithm is used also during the lifetime of a robot, e.g. when it is in motion. This is to make it able to adapt to varying environments. The last one is called Evolutionary Synthesis, and is by many considered as the next big thing in this field. It is about automating the process where you design and build autonomous robots. It should be done without human intervention, given just the specification of a task, and the final goal is that the robots should end up better than what an engineer could make. However, most evolved robots today are still inferior to the ones designed by humans[19], and this way of doing it is therefore the least developed one. From this we can see that there is still a lot of research that needs to be done before this field is fully matured.

2.2.1 The Reality gap problem

One of the major challenges in Evolutionary Robotics deals with avoiding something called the Reality Gap. When training and/or evaluating a system in a simulator, a solution that seems optimal when ran there, can work poorly or not at all when transferred onto the physical robot. This is due to differences in the simulated and physical environments, leading the evolution to match the specifications of the simulated world instead of the physical one. One may think that this problem would disappear by simply removing the simulation phase, and instead run the evolution phase directly on the robot, but there are several reasons why this is not feasible[14]: The first reason is that it is too time-consuming: Evolving several populations with several individuals in a simulator is much faster than doing it on the real robot. This gives the opportunity to run a higher number of both populations and generations, resulting in one being able to address more complex tasks. The second reason is that individuals with poor performances, due to randomness can in fact destroy the robot by e.g. colliding with a wall. One last reason worth mentioning is general "wear and tear" of the robot, i.e. the robot parts can get damaged and break due to them being worn out. Another problem is to always restore the environment to the initial conditions between trails without any human interference. In a simulator, this is of course a non-issue. Because of these problems, the research nowadays is more focused on minimising this gap effect, rather than trying to avoid it completely, i.e. to get similar results both when doing simulations and on the physical robot, according to [19].

Possible solutions

Dynamical properties such as e.g. friction, gravity and inertia are modelled quite well in software libraries that has been developed over the last years. These physics-based simulators are able to simulate movable robots and their environments faster than in real-time on normal desktop computers, which has led to a wide-spread use by most ER researchers today. In fact, most of the work can be done using this kind of software[14], but in spite of that, even these kinds of simulations will have a Reality Gap. For example a problem that occurs in the simulator, is not necessary a problem in the physical world, or the other way around. Therefore, there are several techniques one can use to minimise this problem, described in the following paragraphs.

A simple method is to add random noise to the sensors and to the end position of the robot in the simulator[14][37]. This hinders that the solutions found during evolution are solely based on the peculiarities in the simulation model. A second technique one can use is called minimal simulations. The essence in this one is to accurately simulate only the characteristics that are important to get the desired behaviour[36], while the other characteristics are randomly varied across several trials

on the same individuals in the populations. This ensures that the evolved individuals are independent of implementation aspects, and only dependent on these important characteristics. To get some robustness out of it, it is necessary to also vary these important characteristics a bit, although not to the same extent. Unfortunately, a disadvantage with this method is that it can be hard to find the characteristics that cause the desired behaviours in advance. A third method can in fact be to directly evolve the solutions on the physical robot[25][48] [74], but this has not gained a lot of popularity due to the previous mentioned disadvantages with it. An alternative that can limit the number of physical evaluations is to first evolve in a simulator followed by further development on the physical robot afterwards[48] [56].

Another approach used to reduce the Reality Gap is called simulator tuning. In [6], Bongard and Lipson conduct an ER experiment using this method where the problem is treated as a system identification task. Two general approaches are shown in Figure 2.2 on page 13. The idea is that between the robot simulator and the target robot there should exist hidden differences which must be automatically uncovered and included into the simulation. The aim of the experiment is "to automatically refine the simulator sufficiently that controllers evolving in it cause the target to produce similar behaviour to that seen in simulation", i.e. crossing the Reality Gap. This means that the simulator is evolved itself, and can allow for modifications to the virtual robot's body, it's actuators and sensors, it's simulated environment as well as to the physical parameters of the simulation. Due to the increasing development and availability of physics-based simulators in recent years, the ability to evolve a virtual robot's morphology and surrounding environment in the simulator has become an uncomplicated task.

Figure 2.3 shows a more specific algorithmic flow of this task, where the the target system is a quadrupedal robot with an articulated body, i.e. a robot with joints, with sensors and actuators as well as a neural network controller that connects the sensors and motors. The target robot is usually a physical version of the simulated robot, but in this paper the target used was another simulated robot identical to the first one. The only difference was in morphology, missing one of its lower legs. Experiments discovered a significant drop in fitness, showing a clear gap between the two simulated robots. To work out these non-linear differences and to modify the original simulator enough to accurately reflect them, an estimation-exploration algorithm was used: While the exploration phase is evolving controllers for the target robot, making it move, this resulting behaviour is used to refine the simulator. At the moment, the researcher must select the environmental or morphological characteristics differing and affecting the transferred behaviours the most, manually. The features selected in the described experiment were mass distribution and sensor timing, due to them being the most critical ones for the walking behaviour. To make the algorithm detect these differences, feedback from the target robot sensors was used to modify the default robot simulator and it's mass and sensor

parameters. An experiment of 50 runs was conducted, and the algorithm was able to infer most of the sensor time lag characteristics correctly after only a few trials. It was less successful however, working out some of the body parts characteristics, and they claim that this is because some features can be harder to observe than others when using only sensor feedback. Despite this, after only a few trials using the algorithm, the researchers were able to get only a statistically insignificant decrease in travelled distance between the two robots. This shows that even though the simulator does not describe the target robot perfectly accurate, it makes the transferal of evolved controllers from simulation to target automated, making this an interesting approach for dealing with the Reality Gap.

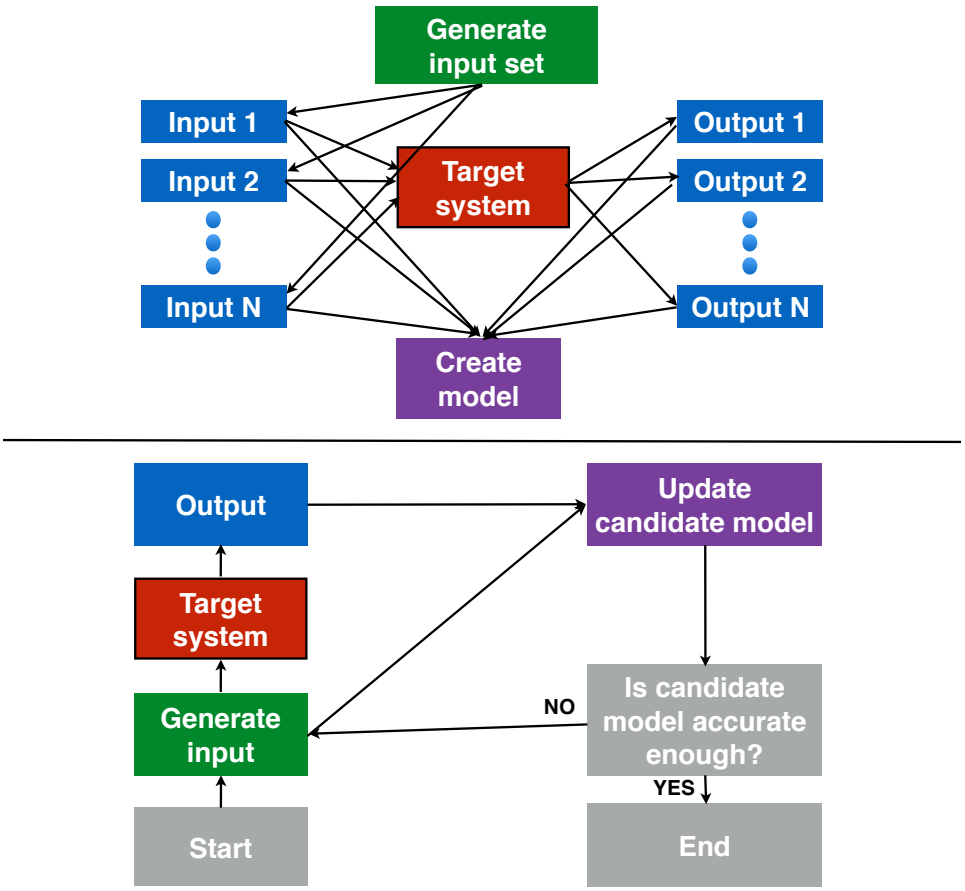


Figure 2.2: Two general procedures for system identification: The top one shows a so-called "batch method", in which one first generate an input set of data which is sent to the hidden target system. This "black box" system then generates a set of outputs corresponding to the inputs, and finally the model of the system is constructed using these resulting input/output pairs. The bottom one shows an iterative method, where one continues to make the input/output pairs for updating the system model until it satisfies some precision requirement.

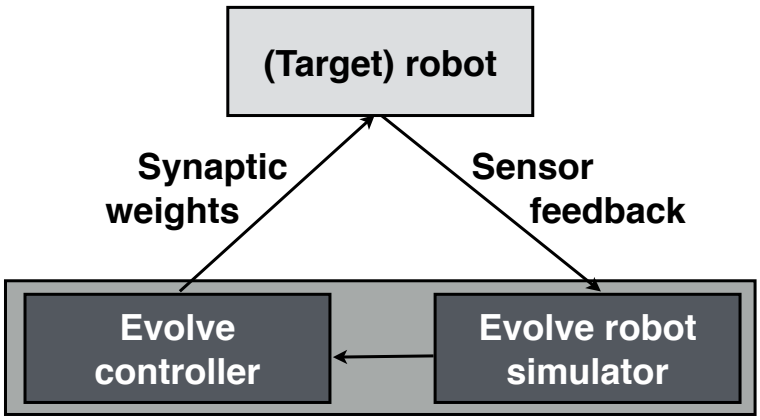


Figure 2.3: System identification task for Evolutionary Robotics. The target system is a robot with an articulated body with e.g. a neural network as controller, and the robot is usually physical. The task is to use e.g. sensor feedback from it to detect hidden, non-linear differences between behaviour in simulator and on the target robot. These should be included and modelled in the simulator and the aim is therefore to make the physical robot produce a similar behaviour to that observed on the simulated one.

2.2.2 The Transferability Approach

Another proposed method to cross the Reality Gap is called the Transferability Approach[44]. While this method also aims to improve the accuracy of the simulator, it does not modify the simulator nor its model itself. As the evolution goes on, the idea is that the solutions found will take advantage of poorly modelled features in the simulator not existing in the real world. Examples can be weights and behaviours of robot actuators, as well as friction and other forces. But improving simulators can only work to a certain degree; one can never have a completely accurate one, and complex simulators that simulate e.g. fluids can in fact be slower than reality. Instead of trying to modify the simulated world to better represent the real world by trying to fix these inaccurately represented features, the evolutionary optimisation algorithm itself is made aware of the simulator limitations. The purpose of this is that the system can identify and avoid solutions it can not simulate correctly. The authors in [44] has described these limitations using a transferability function, i.e. a function describing how well solutions found in simulation performs when transferred to reality. Since the best solutions found in simulation probably takes advantage of poorly modelled and unrealistic features and as a result will not transfer well to reality, one can expect to see a trade-off between fitness in simulation and transferability. The function is for this reason added as another objective to the evolution process, making it an Evolutionary Multi-Objective Optimisation (EMO) problem.

In [52], the transferability function is defined as "a function that maps, for the whole search space, descriptors of solutions (e.g. genotypes, phenotypes or behaviour descriptors) to a transferability score that represents how well the simulation matches the reality". In other words; the purpose of the Transferability Approach is to use it to find solutions that transfer well from where the evolutionary learning is conducted; usually in a simulator, to where the final solution is actually going to be used, e.g. on a physical version of the robot. To make the function, there are two factors that needs to be decided:

1. A similarity measure, i.e. what to compare (function output).
2. A solution descriptor, i.e. what to map the function from (function inputs).

The similarity measure can be highly problem- and simulator-dependent, and for a bipedal locomotion robot, the most intuitive measure is probably the walked distance covered during the evaluation period, i.e. comparing fitness values. Other similarities can be the contact time between the legs and the floor, as well as the robot orientation, angular joint positions or the trajectories of the Centre of Motion (COM), all measured at regular intervals. The other thing one needs to consider is a solution descriptor, representing the optimised solutions in some way. This is what one needs to map the transferability score from. Ideally, the best way to map this kind of function for a walking gait optimisation problem will be to map from the genotype input space. For example in [44], the genotype is represented by only two real gait parameters, resulting in a non-complex, three-dimensional transferability function, effectively explaining how the difference in fitness values between simulation and reality changes for different gait parameters. An example of a transferability function that maps from a genotype space consisting of two gait parameters $p1$ and $p2$ to a transferability score can be:

$$Transferability_{score} = Transferability_{func}(p_1, p_2) \tag{2.1}$$

, where the transferability score is the difference in fitness values between simulation and reality, and p1 and p2 are the two gait parameters in the genotype describing the solutions.

Unfortunately, the chance of having a genotype with only two parameters is quite small when dealing with robots such as humanoids, increasing the chance of getting a highly non-linear or complex transferability function mapping. The reason is often due to their complex dynamics, making the number of genes in the genotype quite high.

This challenge regarding the mapping process can be found in Figure 2.4 on the following page. As one can see, the ideal way is a mapping from a genotype representation; in this experiment the gait parameters used to alter the walk, to a transferability score that tells one the size of the reality gap, i.e. how big the difference in fitness is between the two environments.

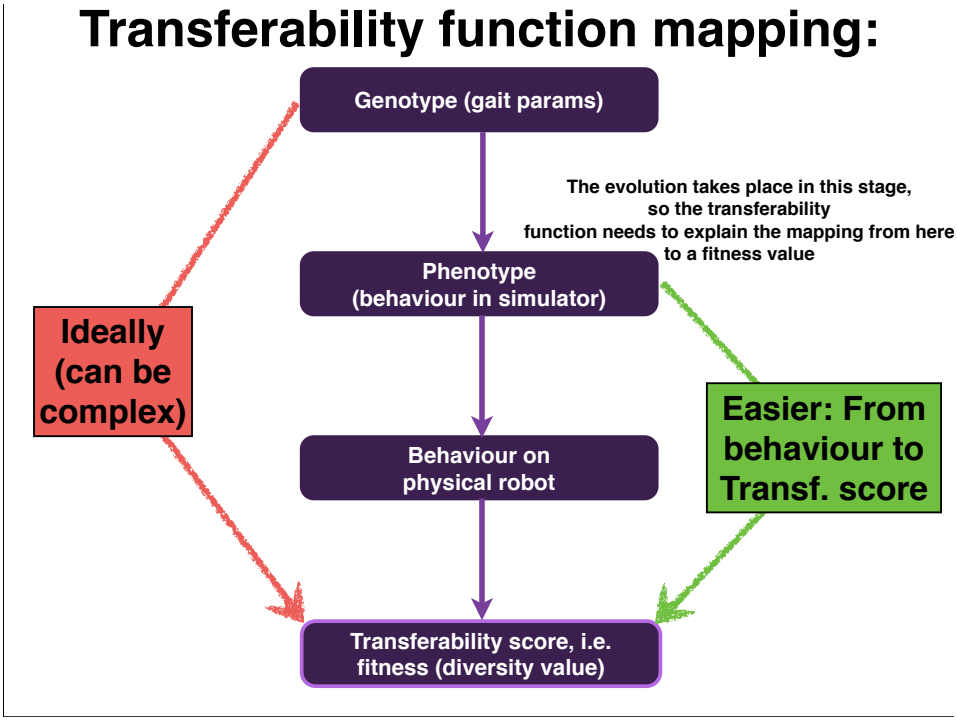


Figure 2.4: The mapping of a transferability function. Ideally, one wants the function to tell how well the evolved solutions transfer to reality, i.e. the transferability score by using the genotype representation, which can be e.g. gait parameters. Unfortunately, this mapping can be highly complex. The solution is then to map the function using e.g. a distinct behaviour in the simulator.

In order to make the function less complex when having a genotype with a large number of properties, an alternative can be to use a mapping from a phenotype or behaviour space: We want to find out if we can expect similar behaviours from the real robot compared to the observed behaviour in the simulator. The choice of a behaviour input space can be highly problem-dependent, but regardless of this, learning the transferability function is done by transferring solutions found in simulation to the real system. Since one can not transfer all the solutions found, the idea is that by using regression techniques on the transferred ones, the resulting function will make good transferability approximations for all solutions in the whole search space. An explanation of the whole approach is found in Figure 2.5, where the approximation can be further improved by continuously transferring newly evolved individuals.

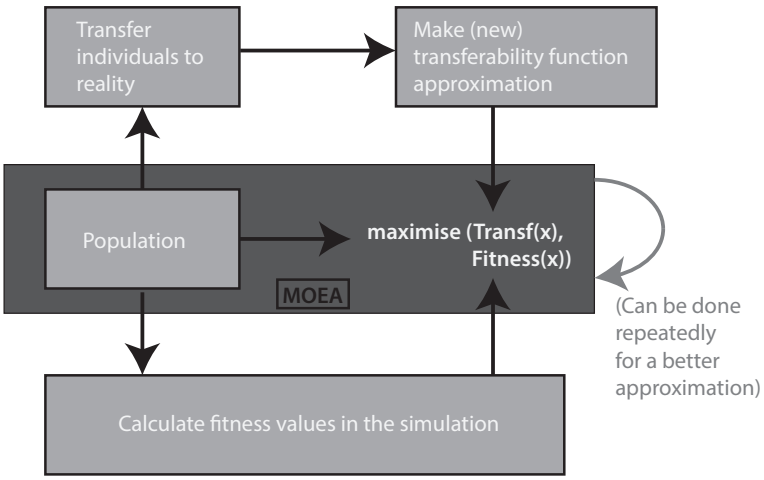


Figure 2.5: Multi-objective optimisation of the two objectives fitness and transferability, i.e. how well the simulation matches the reality. This transferability function approximation is improved by transferring evolved individuals from simulation to reality.

2.3 Humanoid robots and the Nao platform

This section will be about humanoid robots in general, as well as the Nao robot. It will concentrate on what they can be used to, in addition to advantages and disadvantages of using these platforms.

2.3.1 Humanoid robots

A humanoid robot is essentially a robot with the body shape of a human body and like the Nao, it is often also supposed to behave like a human, e.g. physically with bipedal locomotion, i.e. walking with two legs. Other uses can include human interaction, e.g. using oral communication. According to [15], "Over the last decade, the number of humanoid robots developed for research has grown dramatically, as has the research community". Therefore, only time will tell if humanoid robots will be one of the preferred types of robots in the future, competing with both human labour as we know it today, as well as more specialised robots. Humanoid robots intend to mimic aspects of humans like shape, but also behaviour in varying extents. They range from small robot-heads with human-like capabilities such as sensing, e.g. using computer vision, all the way up to robots as big as humans with two legs intended to walk using bipedal locomotion. The reasons for making these types of robots are many, where examples can be both general-purpose mechanical workers, entertainers and also a platform for testing out theories from neuroscience and experimental psychology[15].

One thing worth mentioning is that these kinds of robots offer a great

platform for doing research on interaction between humans and robots as well as integrating them into environments that today are dominated by humans. Another advantage of humanoids that use legs to walk compared to e.g. robots with wheels is that they can be used to walk in rough terrain and soft soil. Since these areas cover over 50 percent of the earth, the usability of these robots can be greatly increased. Unfortunately, the energy consumption and the robustness of legged locomotion humanoids are inferior to the wheeled ones[66]. This is because they need a lot of actuators to move their legs properly, making them quite heavy. They are also neither particularly fast nor easy to build. This means there are still a lot of research that needs to be done in this field.

Examples of humanoids

Some humanoid robots are listed in Table 2.1. As one can see, there are various kinds of humanoids being made, with a varying price range and availability. The most affordable ones are so-called mini-humanoids which are significantly smaller than ordinary humans, where others intend to mimic the body of a fully grown person. While most of the robots are used primarily for research, the Topio made by Tosy is for example designed to play table-tennis against humans. The Nao and DARwIn-OP robots are more described in the following section.

2.3.2 The Nao robot

Overview

The Nao platform consists of a humanoid robot called Nao. It is made by a French robotics company called Aldebaran Robotics and is commonly used by universities, schools and laboratories all over the world for research and education. The robot is a 25-degrees-of-freedom (D.O.F) bipedal robot with a weight of 4.3 kg and a height of 58 cm. For an overview over its joints and motors, see Figure 2.7 on page 21. It is a humanoid which is programmable, and its development began already in 2004. The robot comes embedded with an operating system based on natural interaction and emotion, which makes it able to talk and respond to speech. It also has a software development kit (SDK) that supports many programming languages, such as Java, C++, Python and .Net as well as compilation and debugging tools[2] for both Windows, Linux and Mac OS. In 2008, an academic version of it was released to educational institutions and laboratories, with the purpose of being used for research. The Nao Next Gen was publicly released in 2011, and according to [23], an upgraded version with support for multilingual speech capabilities is going to be released sometime this year. Two cameras located in its eyes, gives the creature the ability to visualize it's surroundings in three dimensions. It also has touch sensors, in addition to four directional microphones, making

Name	**Mass**	**Height**	**D.O.F.**	**Gait speed**	**Main purpose(s)**	**Price**
NAO H25 Evolution V5 (2014)	4.3 kg	58 cm	25	7 cm/s (11 cm/s [47])	Education and research, RoboCup Standard Platform League (SPL)[13]	8.000 USD
DARwIn-OP (2015) [34]	2.9 kg	45 cm	20	24 cm/s	Open research and development, RoboCup Kid Size League	12.000 USD
HRP-4 (2010) [33]	39 kg	151 cm	34	56 cm/s	Development of domestic helper robots in Japan	~300.000 USD
Reem-C (2013)	80 kg	165 cm	44	0.39 m/s	Research	~350.000 USD
Honda ASIMO (2014) [32]	50 kg	130 cm	57	75 cm/s (1.94 cm/s running)	Robotic research, springboard for other Honda project development	Not for sale
Atlas (2013) [20]	150 kg	188 cm	28	Not listed	Search and rescue tasks	Not for sale
TOPIO 3.0 (2010)	120 kg	188 cm	39	N/A	Table tennis (AI)	Not for sale
Romeo [3]	40 kg	140 cm	37	Not listed	Research on assistance for elders and/or disabled people	Not released yet

Table 2.1: A table with info about some known humanoid robots.

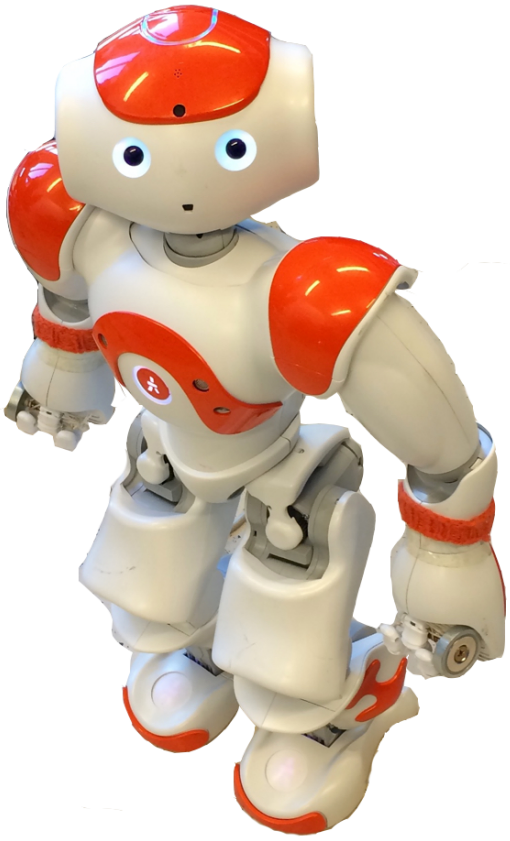


Figure 2.6: The Nao humanoid robot.

word of mouth a possibility. An advantage of the Nao is that it is very robust, with one key benefit being the ability to recover after a fall, using features like an inertial measurement unit that tells the robot if it is sitting down or standing up.

RoboCup

The most known use of the Nao is maybe its role in the RoboCup[24], famous for its football competition. The RoboCup project's ultimate goal is by 2050 to beat the human world champion team in football by using autonomous humanoid robots[75]. As a part of this project is the Soccer Standard Platform League (SPL), where the Nao is the robot currently used by all teams. The purpose of using identical robots is to focus on advances in algorithms for autonomous robots rather than the development of new robots and hardware[75], which differentiates this competition from other robot contests around the world, including the other RoboCup leagues. The robots are fully automated during the games, meaning there are no interventions by humans or computers, neither remotely nor hands-on.

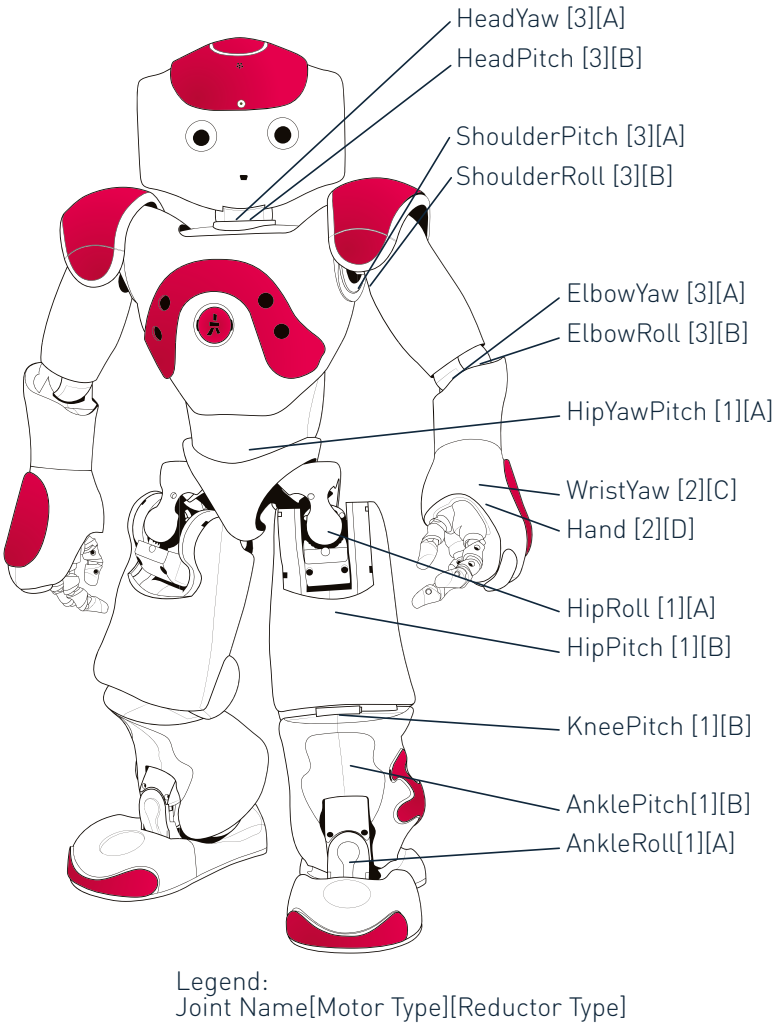


Figure 2.7: The robot's joints, motors and gear reduction ratios, where motor type 1 is the most powerful one[57].

The number of competing teams vary from year to year, but twenty or more teams have been the standard the last years. Another interesting fact is that the size of the football field has since the introduction of the Nao increased from 4.4 times 6.8 metres to 6 times 9 metres this year. The number of team members has also grown from two in 2008 to five plus a coaching robot in 2014, significantly increasing the complexity[13] of the contest.

DARwIn-OP

A robot comparable to the Nao, at least in size, is the DARwIn-OP[34], from now on mentioned as the Darwin and pictured in Figure 2.8. It is made by the the Korean robot manufacturer Robotis, and is developed in collaboration with the American universities Virginia Tech, Purdue

University and University of Pennsylvania. It is a humanoid robot made for development and research in fields such as Artificial Intelligence, bipedal locomotion, computer vision, linguistics and humanoid robots themselves. But unlike for the Nao, the hardware platform is open to modifications, i.e. open-source, encouraging people to not only modify the controller software, but also the hardware. Due to its modular design and publically available CAD files, one can even manufacturer and put together the robot by oneself[59], although one still needs to purchase the required actuators. A point worth mentioning is that all the information about hardware modifications is required to be put online for everyone to use, stating the focus on the open source even more clearly. This distinguishes the Darwin from the Nao in great fashion, where you need to buy the robot from the manufacturer and where the research is solely focused on improvements in robotic software and algorithms.

The Darwin has like the Nao also been used in the RoboCup competition with success, winning the RoboCup Kid Size League for several years [60]. When it comes to research, it has recently been used in several relevant experiments, where examples are [50] and [67]. The first paper describes different biped locomotions and other walking behaviours generated by rhythmic movements using Central Pattern Generators (CPGs). The gaits were developed in a progressive way, i.e. basic elementary motions were combined to finally get different bipedal locomotion behaviours. Tests on a simulated and a real Darwin robot made the robot able to follow a ball using these different gaits. The second paper describes an approach to automatically generate a bipedal locomotion controller for the Darwin in a physics-based simulator. They utilised an EA variant called Genetic Programming (GP), in a combination with feedback from the robot's own sensors. This resulted in a faster gait than the default one on a flat surface, as well as making the robot able to walk in a restricted, tilted environment. An important discovery was that the sensory feedback was of particular importance when adapting the robot's gait to the sloped ground.

2.4 Bipedal locomotion

2.4.1 Traditional approaches to walking methods

There are several challenges to consider when evolving walking behaviours on humanoid robots, and there have therefore been developed a number of distinct approaches that focus on different aspects of these problems. The researchers in [28] argue that one should use an Evolutionary Algorithm (EA) to optimise walking gaits. This is due to their strength and robustness in global search, as well as their ability to deal with imprecise models of robots, as can be the situation with very complex humanoids. Since the algorithms can easily be run in parallel, in addition to cope with multiple objectives and constraints, the effectiveness in these

Figure 2.8: The DARwIn-OP open-source humanoid robot[34].

kinds of optimisation problems is superior to many other approaches. A final argument is that since EAs are biologically inspired optimisation, there is a chance that the evolved behaviours works well in a biologically context as well, such as in the real world.

2.4.2 ZMP- and Inverse Pendulum-based approaches for stability

When it comes to the generation and control of walking gait patterns, the works can mainly be divided into two groups: The first approach is based on the Zero-Moment Point (ZMP)[38] of the robot, and is in need of specific robot dynamics such as the mass, the location of the Centre of Motion (COM) of the torso as well as link inertias to prepare the walking patterns. The ZMP is a point where the contact between the foot and the ground does not produce any horizontal momentum. In other words, it is the point where the combined gravity and horizontal inertia forces equals zero, and is therefore the point where the weight of the entire foot should be placed. This ZMP is critical when it comes to the dynamic stability of the robot, i.e. not walking in an unstable way or falling over, more than it is concerned about the gait speed. A key assumption is that if the trajectory of the ZMP is kept inside the area that the support foot can reach during the walk, the locomotion should be physically stable, making the robot not able to fall over. The approach also assumes a more or less flat surface, having a high enough friction to prevent the feet from slipping, and is one of the most

common approaches in the research field of humanoid robotics[77].

The other approach on the other hand only requires the knowledge about simple dynamics such as the location of the total COM and the total angular momentum of the robot, and is called the Inverted Pendulum Approach[72]. To make up for the lack of specific information about the system structure, it is dependent on some kind of feedback while walking. A problem with using feedback to generate stable gaits is that it alters the original task-dependent foot placement, making it a problem to walk on e.g. stepping-stones. A solution to this problems has been proposed in [41], where the authors have come up with a method that merges these two approaches together, utilising that the ZMP-based one does not have this problem with arbitrary foot placement, while still allowing for a pattern generator based on a simple Inverted Pendulum model. A more detailed explanation of this method is described in section 2.4.4.

2.4.3 Other approaches

Another approach can be e.g. to evolve gaits using a Central Pattern Generator (CPG), which is described in more detail in section 2.5.1 and 2.5.2. This kind of approach is usually also concerned about stability. However, there are also other ones concerned about e.g. locomotion speed, energy consumption, minimisation of torque change as well as the smoothness of the walk. Speed is maybe the second most important factor after stability, while focusing on energy consumption has the potential to make the gaits more human-like, according to [28].

A recently published paper that focuses on speed, i.e. fast walking is described in [64]. Unlike for approaches based on the Linear Inverse Pendulum model, this method allows for a dynamic hip height during the walk. In fact, it focuses on finding and learning good parameters for a model of the hip height motion to make for a rapid walk. The trajectory of the hip height is calculated by using Fourier basis functions, which is in turn used as an input to a CPG, to generate smooth trajectories. In order to make a stable and balanced walk, the Inverted Pendulum Model is utilised. The optimisation of hip height trajectories and gait parameters is done using an Evolutionary Strategy (ES) called Covariance Matrix Adaption ES (CMA-ES), and is mainly focused on walking speed. Experiments on both forward and sideway walks conducted on simulated and physical Nao robots, revealed that they performed better, i.e. walked faster than gaits generated using approaches with a static hip height. A final experiment conducted on the Nao made it able to walk with a speed of 34 cm/s, significantly faster than the default one, found in Table 3.1 on page 54.

2.4.4 The Nao walking method

Since the bipedal walk for humanoid robots is a highly nonlinear and dynamic system, the Nao's walking method uses a simple dynamic model inspired by a technique called the Linear Inverted Pendulum Model briefly mentioned in chapter 2.4.2. An experiment that utilises this technique for bipedal walking on uneven terrain is described in [39]. To solve this dynamic model, one has utilised a technique called quadratic programming[79]. The whole process is explained in more detail in the next two sections.

Paper - ZMP Preview Control scheme

As has been mentioned briefly in chapter 2.4.2, this described solution[41] allows for both arbitrary foot placements in addition to utilising the precise dynamics of a multi-body robot, although the pattern generator is based on a simple Inverted Pendulum model called the Linear Inverted Pendulum Model (LIPM). In this proposed approach, each step is divided into two different phases, distinguished only by the number of feet that is supported by the ground. The double leg support phase, i.e. when both feet are standing the ground, takes up one third of the step time to make for a smooth phase exchange. This exchange is assumed to be happening immediately, resulting in one foot being supported by the ground at all times, just like in the Olympic event of racewalking. The rest of the time is hence the single step support time, where the robot is standing only on one of the feet.

During the support phases, the robot tries to remain its body posture in a vertical position by controlling the support foot's leg and hip joints as in addition to keeping the Centre of Mass (COM) at the same height. This is done by applying a constraint control, i.e. a constraint plane making the mass, i.e. the robot move along it. The result of putting to use this restriction is a simpler, linear dynamics model called the 3D Linear Inverted Pendulum (3D-LIPM)[40] model. A point worth mentioning is that although using a LIPM allows for an omnidirectional walk, it simplifies the robot model to make it linear. It does this by assuming a fixed hip or COM height, although biomechanics has proven that the height of the COM is changing when humans walk. Inverted Pendulum Models (IPMs) on the other hand allows for dynamic COM movements along the Z-axis, but does not work well if one wants the robot to be able to take arbitrary foot placements. This has been mentioned in chapter 2.4.1, and makes it an important fact to keep in mind if using this ZMP Preview Control scheme. The formulas for this dynamics are:

$$\ddot{y} = \frac{g}{z-c}y - \frac{1}{mz_c}\tau_x \tag{2.2}$$

$$\ddot{x} = \frac{g}{z-c}x - \frac{1}{mz_c}\tau_y \tag{2.3}$$

assuming the constraint plane is horizontal, where m is the mass of the pendulum, g is gravity acceleration, and tx and ty the torques around the x- and y-axes. The ZMP position can also easily be calculated using these equations:

$$p_x = -\frac{\tau_y}{mg}, \tag{2.4}$$

$$p_y = \frac{\tau_x}{mg}. \tag{2.5}$$

By substituting them into the two dynamics equations above and rewriting them, one ends up with the so-called ZMP equations that gives the ZMP motion trajectories in the x- and y-plane:

$$p_y = y - \frac{z_c}{g}\ddot{y}, \tag{2.6}$$

$$p_x = x - \frac{z_c}{g}\ddot{x}. \tag{2.7}$$

However, when one wants to generate a walking pattern, the problem is actually the opposite of these ZMP equations: One is simply interested in calculating a motion that controls the COM given the current ZMP trajectory, something that is controlled by the desired footholds and step periods. Earlier methods listed in the paper used to generate the COM trajectories in batches by using a ZMP reference for a specific period of time, making the system more static than dynamic. The result of this was that one either needed to calculate the entire trajectory off-line or couple the trajectory pieces calculated from the different ZMP reference segments together. The challenge was therefore to figure out a system that would generate these adaptive walks online, considering the dynamical state of the humanoid. The solution to this problem was to instead generate the walking pattern using a so-called preview control that works as a dynamic ZMP tracking system. This type of control actually needs to utilise future information when the robot takes a step, because one wants to generate the COM trajectory in a way that makes the resulting ZMP to follow the reference ZMP. This may sound non-trivial, but does not violate any causality laws, according to the authors. They claim that it is like driving a car on a curvy road, where one controls the car's direction by watching forward, looking into the future.

In the support exchange period of a step, i.e. when the support is shifted from the one leg to the other and both feet are standing on the ground, the reference ZMP should experience a step change, and the ZMP should move from the currently supported foot to the other one. The COM of the robot should therefore start moving before this to make the resulting ZMP to follow the reference ZMP. The whole process of the ZMP Preview Controller can be seen in Figure 2.9, where the future input is used to calculate the current output. This is solved by using quadratic programming, which is more described in the next section. In experiments, it was discovered that a preview period of 1.6 seconds was sufficient to

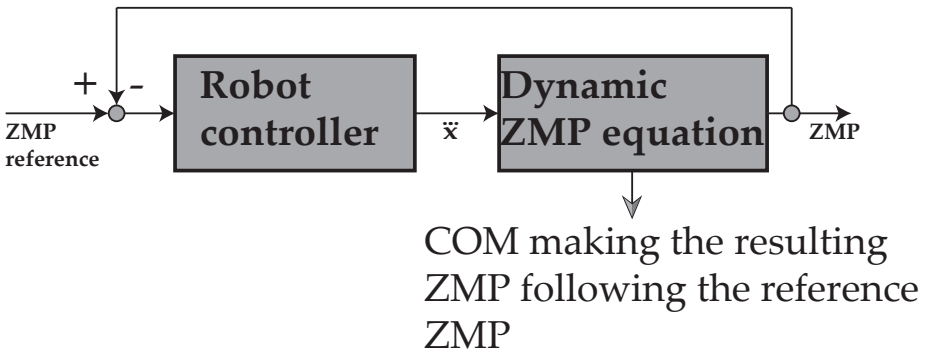


Figure 2.9: Pattern generator ZMP tracking control. The idea is to make the resulting (future) ZMP to follow the the reference ZMP by controlling the COM of the robot.

generate a smooth COM trajectory with good accuracy. The ZMP reference used was designed to stay in centre of the support foot in the single support phase, while moving from the current to the new support foot during the double support phase. To actually generate the walking pattern, solving inverse kinematics was done to make the COM follow the output of the preview controller. The approach was successfully tested on a HRP-2 humanoid robot, with a ZMP error of just 2.3 and 2.6 cm in x- and y-direction, respectively. This can be seen as a quite remarkable result, given the huge size and weight of the HRP robot series, mentioned in Table 2.1.

A problem with this approach is if the ZMP error becomes too big, compared to the stability margin that is decided by the geometry of the robot feet: A smaller foot surface is naturally less stable than one that covers a larger area. A big error can make the robot fall over, but by using the preview controller again, it can be reduced: If one compares the COM trajectory of the multibody-model with the one from a simpler model with no error, e.g. by modelling a cart on a table, the expected ZMP error can be found. This information is then stored in a buffer, and is used after a delay time equal to the preview controller length. One can now calculate a compensation for the error in the preview controller by utilising the future ZMP error. A resulting experiment was therefore conducted, and showed an improved ZMP pattern where the multi-body model ZMP was able to follow the cart-table model in a much better way. The maximum ZMP error was now reduced to only 1.2 and 0.4 cm in x- and y-direction, respectively. A shorter preview period of only 0.75 seconds was also found to be sufficient, because now only a small compensation was needed.

Paper - Solving the ZMP Preview Controller using Quadratic programming

In the previous section, the actual walking pattern generation is done by solving inverse kinematics. In [79] however, the ZMP Preview controller uses a technique called Model Predictive Control, solving a quadratic program instead. In general, a quadratic program problem is about optimising, i.e. maximising or minimising a quadratic function of distinct variables, which have their own linear constraints. More specifically for this chapter is the program below:

At every time step, kT, the quadratic program needing to be solved is given by the formula:

$$min \sum_{i=k}^{\infty} \frac{1}{2} Q(z_{i+1} - z_{i+1}^{ref})^2 + \frac{1}{2} R \dddot{x}_i^2, for \dddot{x}_k, \dddot{x}_{k+1}, ... \tag{2.8}$$

where the ratio R/Q allows one to use the future tracking reference positions given by the reference ZMP, zref, to balance the minimisation of the jerks, given by the triple-derivated x. In other words; one simply wants to minimise the ZMP error, which is the difference between the future ZMP reference position and the actual next ZMP position, by keeping the ZMP position inside some limits depending on the horizontal feet positions on the ground. This position is naturally dependent on the current support phase, and the jerk minimisation is done while maintaining the ZMP position as close as possible to the reference ZMP at every time step, kT.

The novel idea of Model Predictive Control described in the paper is to just execute the first time step of this trajectory, i.e. from *kT* to *(k+1)T* to measure the actual state, i.e. the COM positon and velocity. A new trajectory is then recalculated using the same quadratic formula, saving this current measurement to use as a state feedback. In fact, solving the formula over just a finite time N was discovered to be sufficient:

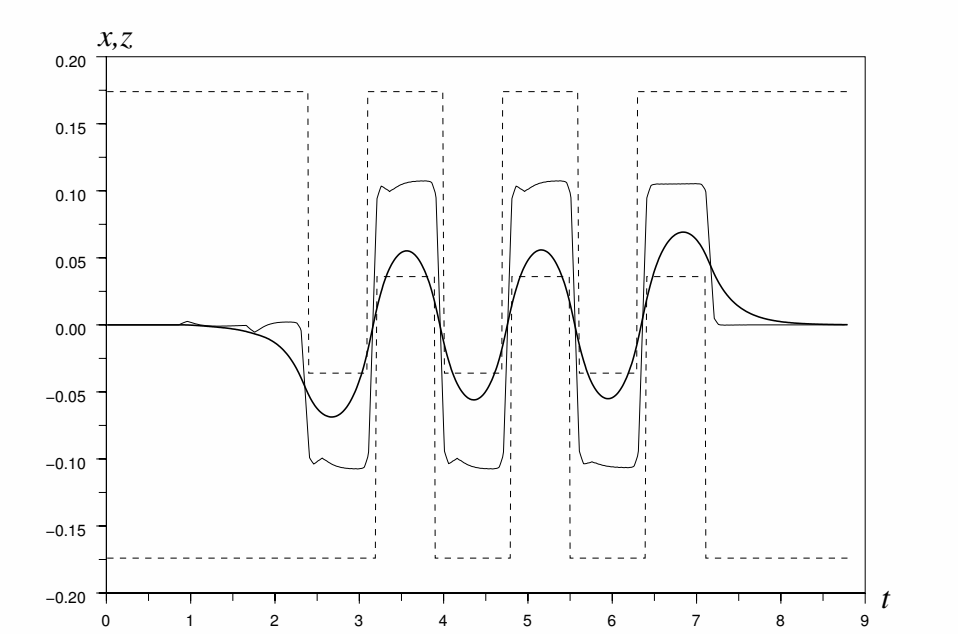
$$min \sum_{i=k}^{k+N-1} \frac{1}{2} Q(z_{i+1} - z_{i+1}^{ref})^2 + \frac{1}{2} R \dddot{x}_i^2, for \dddot{x}_k, ..., \dddot{x}_{k+N}. \tag{2.9}$$

An example of this approach can be seen in Figure 2.10. The approximated ZMP position stays inside the admissible positions using both a simple dynamic model as well as when the approach utilises the whole dynamic model of a humanoid robot (HRP-2), as wanted.

Nao specific

As for the Nao robot specifically, the walking methods described in the two previous sections should give a good indication on how it generates walking gait patterns: It utilises a simple dynamic model called the Linear Inverse Pendulum Model(LIPM)[39], similar to the one in the ZMP Preview

(a) Result using a simple dynamic model.



(b) The same approach as in figure a), but this time using the whole dynamic model of the humanoid robot (HRP-2).

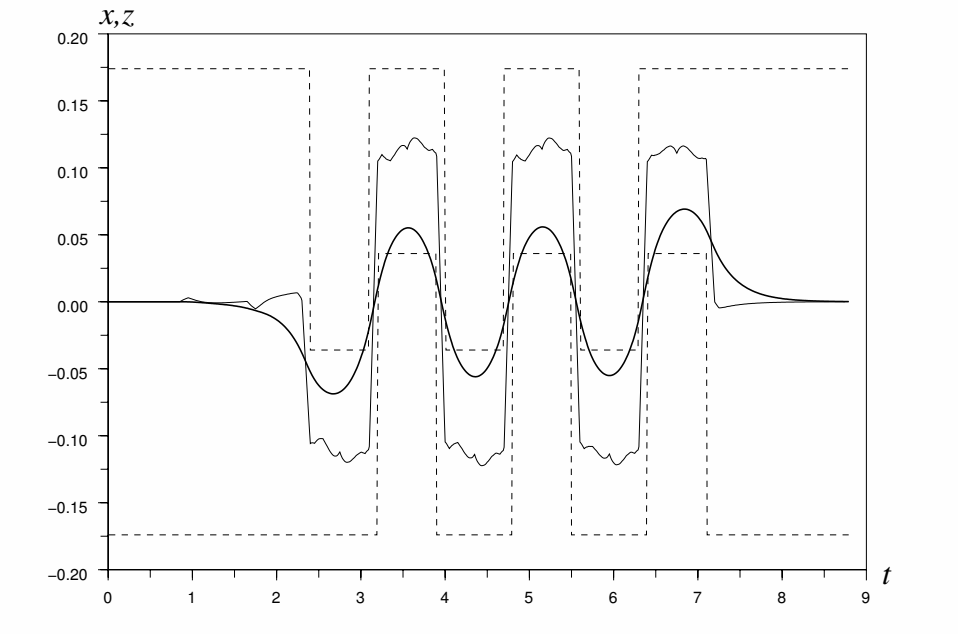


Figure 2.10: The ZMP Preview Controller solved using Quadratic programming[79]: The sideways position of the COM of a humanoid robot taking six steps, as well as the approximated position of the ZMP using equation 2.9. The dashed graph represents the minimum and maximum admissible positions.

Controller approach. This model is however not solved by using its original approach, but instead solves a quadratic program explained in the Model Predictive Control paper. This improved version of the ZMP Preview Control Scheme has been shown to stabilise the walk of humanoid walking robots by trying to minimise the jerk of the COM trajectory. The approach generates stable walks that takes into account the actual robot state by recomputing the walks online, making the walks robust when it comes to dealing with outside impacts. As it e.g.has been proven to work well on the previously mentioned HRP-2 robot, other humanoids such as the Nao should be suitable for this gait generator as well. However in both approaches, the ZMP criterion is used as a stability measurement, and the COM trajectory is determined from the ZMP by using this simple physical Linear Inverted Pendulum Model (LIPM). Although this simple model assumes that the legs of the robot do not have any mass, a successful implementation of a bipedal robot using it as a control system is described in [39]. The researchers here discovered that this support phase dynamics model can be regarded as linear although the legs have mass, meaning the dynamics equation still holds. They also found out that the exchange between the phases can be made smooth by keeping the robot in the double leg support phase for some time, something the Nao implementation also does, in addition to placing the feet using a specified vertical speed. The preview period used to generate the COM trajectory on the Nao is 0.8 seconds, indicating that it uses the improved technique mentioned in the ZMP Preview Controller paper where the preview controller is used again to reduce the ZMP error.

A full walk cycle on the Nao consists of four distinct steps:

1. A double support phase, i.e. the start position.
2. A left support phase, i.e. taking a left step.
3. Followed by a new double support phase.
4. Ending with a right support phase, i.e. taking a right step.

In addition to this, the walk is initialised and ended with a double support phase of 0.6 seconds. As has been before, the walking pattern generator assumes that the height of the COM does not change in any significant way because of stability, i.e. trying to maintain the same height while walking. This is also mostly true for the Nao since it uses the same pattern generator, but to avoid singularities and to make it able to take longer footsteps, the height can be adjusted automatically by the algorithm.

While the Nao marches, the arms can be activated to make the walk more stable. A result of this is that it also improves how the walking looks, and is enabled by default. However, motions of one or both of the arms can be disabled completely or for a given number of time. The arm trajectories, as well as the foot swing and torso trajectories are interpolated using a SE3 Interpolation[82], taking into account the starting points and current speed, ensuring that the trajectories are smooth and that they obey any

speed limits.

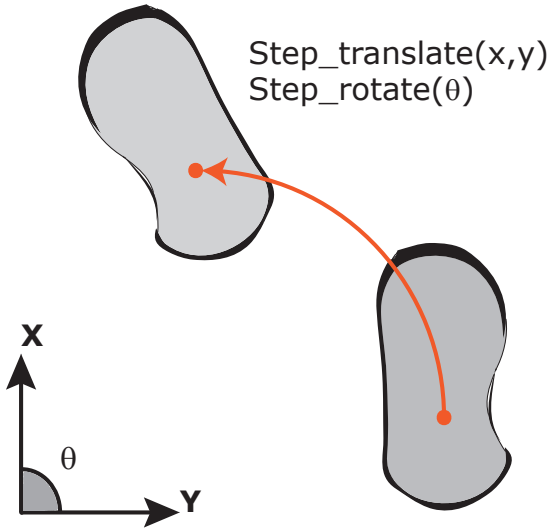


Figure 2.11: The placement of a footstep using the NAOqi Footstep Planner. It consists of a translation along both the horizontal axes x and y, followed by a rotation around the vertical z-axis.

The Nao footstep planner

The Nao's walking method defines a new foot position for one of the feet relative to the current position of the other foot, and the placement is governed by the pattern generator method described in the previous section. The new position is given by a leg translation in both x- and y-direction, followed by a rotation, theta, around the z-axis, i.e. the vertical direction, all corresponding to the other leg. If one uses the built-in walking methods, this footstep planner is utilised automatically, and tries to find the best possible foot placements, i.e. the most stable ones according to the gait parameters specified in Table 3.1. There is also the possibility of planning the robot footsteps directly, either stating just the parameters, or including the velocity of the stepping as well. The planner will in this case clip the specified footsteps to ensure collision-free and singularity-free steps using the maximum and minimum values described in the top three rows in Table 3.1. This also happens in the first case if the specified gait parameters are outside their ranges, when one for example tries to take a larger or wider step than what is possible or stable, or a step that makes the feet collide with each other, potentially making the robot fall over. A footstep placement showing both placement and rotation is illustrated in Figure 2.11.

2.5 Evolutionary Robotics on humanoids and the Nao

This section is about previous research in Evolutionary Robotics, especially for real humanoid robots and the Nao robot in particular. It is meant to give an overview of the history and the standings today. The first described paper is about making the general bipedal humanoid walk more stable by increasing the previously mentioned double support phase, while the next two are focused on improving the walk on the Nao robot specifically, although the first one also should be possible to use on other humanoids as well.

2.5.1 Evolutionary CPG with increased double support time (2011)

A lot of research has been done on stable bipedal walking for humanoid robots in the last few years, despite the high-dimensional complexity of such systems [62] [1] [54] [51]. In [55], Chang-Soo Park et al. tried to make a humanoid robot's bipedal walking more stable by increasing the double support time. At any time in bipedal walk, you are either in a single or a double support phase, and according to [73] and [76], when humans walk; they are in the double support phase around 20 percent of the time. However in previous research, this fact has not been taken into much consideration and has resulted in a double support phase of under five percent most of the time. By knowing this, the researchers had a good motivation for testing out if increasing the time in this phase would help improving the stability. They also used feedback from force sensors attached under the feet to modify motor patterns for dealing with uneven terrain and other environmental disturbances. When the sensors detect any disturbances, they alter the motors in a way that keeps the robot in balance, and thus avoids it from falling over. To measure the effectiveness of the solution, it was simulated in the Webots physic-based simulator[78] using a model of a small humanoid robot called HSR-IX[81], which is comparable in both size and weight to the Nao.

To produce the gait of the robot, a Central Pattern Generator (CPG) was used in combination with an Evolutionary Algorithm. The CPG was not only used to generate the swing motions of the ankles, but also of the arms and the centre of pelvis, to make it more stable. When a humanoid robot's walking speed is accelerating, there is a possibility that it can slip; due to yawing. This is why the CPG also makes the arms move. A common way to create a CPG in robotics is by using a neural oscillator. For optimising the parameters of this CPG, a quantum-inspired EA was implemented[30] [31]. This kind of algorithm is inspired by quantum computing, using e.g. quantum bit representations instead of binary or symbolic representations. The authors claim that the advantage of using it is that it easier balances between exploration and exploitation of possible solutions than traditional Genetic Algorithms. They also claim that using this kind of representation

leads to a quite fast algorithm with regard to finding the global solution in the search space, only requiring a small number of individuals, decreasing the learning time.

As for the experiment, the algorithm was used for optimisation in two individual processes: It was first used to make the neural oscillator produce a longer double support phase by modifying the connecting weights in the neural network, and afterwards the sensory feedback scaling factors was exploited in order to improve the balance of the robot. It was shown that when the centre of pelvis was in an angle such that the double support phase increased, the stability of the robot was improved. In addition when using the feedback from the sensors, they also discovered that the centre of pelvis angles were smaller, which prevented the robot from falling over as well as maintaining the balance.

2.5.2 Automated CPG-Based walking for the Nao (2012)

A lot of research has been done on the Nao, especially for optimising the bipedal walking. There are different approaches and methods to solve this challenge, and Evolutionary Algorithms can be used alone or together with other techniques. One of the simplest ways to attack this optimisation problem is by minimising the number of Degrees Of Freedom (D.O.F.), and therefore minimising the search space used by the EA. An example of doing it this way is described in the this section.

Inspired by biology and based on evolution, two researchers at the Mexican university Tecnológico de Monterrey, decided to use a Genetic Algorithm to solve the bipedal locomotion problem, i.e. to learn a robot with two legs to walk. They decided to optimise the walk of the Nao robot[75], but the solution they came up with, did not require the dynamic model of the robot. This means that the same procedure can be used on other biped robots as well, and can be a good example of how to deal with the problem of specific-purpose ER research. All the research was done in Webots, with the aim of trying it out on the physical robot in a following experiment.

A Central Pattern Generator (CPG) is used together with the GA to generate the gait. CPGs are inspired by neural networks found in animals that makes them generate periodic outputs with aperiodic inputs. They are based on so-called limit-cycle oscillators that are linked together, and was used to generate trajectories for the Nao's motors. For the coupled oscillators to work together, the Genetic Algorithm is used to evolve parameters that will synchronise them well. By doing it this way, the number of parameters to be optimised is kept at a minimum. This means that one will hopefully achieve good results during quite the few number of iterations. Limit-cycle oscillators has in addition to a distinct period, also a distinct amplitude. The result of this is that if the cycle, i.e. the gait of the robot is affected in some way, it should be able to return to its original way of walk after some time. Thus, it should make the robot unaffected by small

disturbances. It was decided to use one oscillator per degree of freedom, and the researchers reduced the complexity of the problem from the Nao's at that time 21 degrees of freedom, to only the ones necessary for producing the momentum needed for walking: Three in each leg; one for the ankle, one for the knee, and one for the hip. Including more degrees of freedom could have helped to make the robot more stable, but at the cost of using more time to get good results.

Some difficulties occurred when the fitness function only focused on the length the robot could walk. In fact, instead of the expected walking, it started jumping to get as far as possible. Another problem was that the robot walked in other directions than straight ahead. To cope with this, the final fitness function gave a reward for moving forward, a small reward for not falling plus a required minimum distance of walking of 20 centimetres. The result was that the robot learned to take small steps, and therefore avoided falling over. Since the walk behaves like an oscillator, it managed to walk the whole training field with no problems just by repeating the learned cycle. The biggest challenge was to make it start walking from a standing position, i.e. to take the first step. The solution they came up with was to start the servos in the legs at the same time as the CPG, and then wait for them to be synchronised.

The optimized walk was after a 16 hour session shown to be about twice as fast as Aldebaran's default engine in the simulator. By adding more degrees of freedom to the search space and using the additional ones for stabilization, it is expected to perform even better. Other research that needs to be done is to extend the robot's abilities to also walk to the side, to rotate as well as an omnidirectional walk. An implementation onto the physical robot as a replacement for the default walk is also something that remains. Some of these things has been done in the research described in the next section, although using a different approach.

2.5.3 Evaluation of Walk Techniques for the Nao (2011)

In [47], two researchers at the University of Newcastle, Australia tried to improve the Nao's walking speed by comparing three distinct optimisation algorithms with different parameters. They also used different fitness functions, including one based on efficiency and one on speed. The algorithms they tested were an Evolutionary Hill Climber, a variant based on Reinforcement Learning plus one based on Particle Swarm Optimization. All the experiments were done in Webots, and the goal was to implement only the best combination on the Nao robot afterwards, to minimise the pressure on the physical robot. The path they tested it on included both forward, sideward and diagonal components as well as an omni-directional backwards spin. The walk engine they used is described in [35], and is similar to the one provided with the Nao.

To measure the performance, they took into account both speed, cost

of transport(efficiency) and the stability of the gait, i.e. how you walk, and took the mean of them over several trials. They found out that the Reinforcement Learning algorithm, called Policy Gradient Reinforcement Learning[43] performed better than the other two, in addition to having less variation between the trials. It also made the robot fall less often, signifying more stable gaits, coming from how the specific algorithm works. By averaging several nearby positions to estimate the gradient, it avoids fast areas that are unstable or inconsistent. The Evolutionary Hill Climber on the other hand, could very well try to explore such areas. The fitness function that gave the best results, was the one concerning efficiency. It even improved the walk speed more than the speed-based fitness function. The reason is because of the Nao's high energy use; walking the test route faster, simply means using less energy. This analogy can be drawn to humans, as we also have a high basal energy burning[12]. Also, a more stable gait results in less unnecessary energy going to waste. The researchers also decided to expand the walk parameter space to include the stiffness of each robot joint, as it has been shown to improve the performance[45][46]. The stiffness for the left and right legs was the same, but the joints in each leg could differ. The result was that the walk became slower, but at the same time more stable and more efficient.

After the satisfying results in the simulator, the optimized walk was directly applied to the physical Nao to improve the default walk engine, delivered by Aldebaran. The average speed increased by about 57 percent, from 7.0 cm/sec to 11.0 cm/sec, while the efficiency was improved by one third, from 16.8J/Nm to 11.8J/Nm. The robot also never fell over, neither in the simulator nor on the physical one.

2.5.4 The standings today and the future for the Nao and other humanoid robots

According to the author of [70], "it is expected that in the future vehicles adopting locomotion through artificial legs may become an efficient transportation mode, that can compete with other classical transportation means." But nevertheless, in this stage of the development and as you can see throughout this chapter and thesis, there are still some features that needs further improvements. The most significant of these can be the big drawback when it comes to the energy use, especially for walking: When one compares it to robots with wheels as well as the efficiency of living animals. Several optimisation techniques for these robots have been proposed, like choosing the most efficient gait and adapting to the environmental conditions such as the terrain. In addition to these, other methods can also be found in the design and construction phases of the robots.

Another thing that has not been delved into in this paper, which is important when it comes to the future of humanoid robots is about people getting comfortable having robots around. The CEO at Aldebaran Robotics

said in a stage performance that their goal is to make emotional robots that can connect with people, as well as help them against loneliness. He therefore believes that humanoid robots, like their Nao needs to have a friendly look with good interaction and dialogue systems that mimic human behaviour. Features making robots understand not only speech, but also emotions from body language is getting essential in this work of getting people comfortable having them around. If the robots themselves also have an expressive body language, Aldebaran believes they will have an advantage over their competitors.

As for the research part, most of it is problem-specific, and can not be easily reused by others. To improve this situation and move from proofs of concepts to more solid knowledge, the author in [19] suggests to import the best practices from the overlying fields of robotic engineering and experimental sciences.

Chapter 3

Implementation

This chapter is about the work towards an integrated evolutionary learning platform capable of testing out various Evolutionary Algorithms. The first section is an investigation of the current available software packages, and is intended to give an overview of their possibilities, as well as giving an indication of what is needed to be implemented to make the platform and robot ready for use in experiments. The next sections are more implementation- specific, including an overview of the reasons for choosing a capable learning module, robot control system, simulator and Evolutionary Algorithm for the experiments.

3.1 Capabilities of current available software packages

The standard software packages for use with the Nao robot and platform is currently made up of a Software Development Kit (SDK) that is included when you purchase the robot in addition to a physics-based simulator that intends to mimic the Nao's behaviour in the real world. To make it easy to both simulate, interface with and integrate the robot in an evolutionary learning loop, it is desirable to not make these different parts separated of each other. To make them all parts of the same program, one at least in theory will minimise the need for writing the same code all over again, and instead make the different modules interact with each other in a reasonable way. An example of this is to make the robot controller available for both the evolutionary learning platform as well as for testing. Another desirable criteria can be to make the way of communicating with both the simulated and the physical robot as similar as possible.

3.1.1 Simulation

If you do not want to create your own simulator environment or embed the Nao model in your own simulator, there are currently two options for simulating the Nao, namely the commercial Webots and the open SimSpark.

Webots

Webots[78] is a commercial mobile robot simulation software developed by Cyberbotics Ltd. It has in recent years become one of the most used three-dimensional robot simulators, especially for applying evolutionary techniques to robots and humanoids. Webots has been in continuous development since 1996, and uses a physics engine called the Open Dynamics Engine (ODE) library[68] to simulate rigid body dynamics. With out-of-the-box support for many of the most well-known available robots today, as well as including the possibility to model and simulate your own robots, it is currently being used by researchers from over a thousand different universities and other research centers[21] around the world. It comes with an API for accessing and modifying pre- or self-made nodes, such as robots and physical elements, in addition to a Nao-specific version.

SimSpark and other open source alternatives

SimSpark[80] is a simulator for simulating multi-agent, i.e. several robots in a three-dimensional environment. It supports building applications with the open-source application framework called Spark, and is used as the official RoboCup 3D simulation server in the RoboCup 3D Soccer Simulation League that uses the Nao robot. It also utilises the ODE library, and its primary purpose is to provide a generic and flexible simulator for a variety of simulation tasks.

Another alternative can be the Gazebo simulator, which according to the ROS wiki page[26] soon will get a plugin for simulating the Nao.

3.1.2 Interfacing and integration of a robot control system

This section describes the two most common ways of controlling the Nao. The goal is to find a robot control system that can be implemented into an integrated evolutionary learning loop, possibly interfacing with the simulator in addition to other and self-made modules.

Official API - the NAOqi Framework

The Nao robot runs and is controlled by an embedded operative system called NAOqi, and the official way to communicate with it is by using the official Application Programming Interface (API) called the NAOqi Framework. In fact, all of the Aldebaran robots utilises this programming framework. It is very powerful, and incorporates features like parallelism, resources, synchronisation and event handling. It allows for creating distributed programs run remotely from a computer or locally on the robot. It is cross platform as well as cross language, making it possible to create software on different platforms using languages such as C++ and Python. To develop e.g. robot controller software, one can either use Aldebaran's software suite called Choregraphe, which supports Python and has pre-programmed modules, or start from scratch using the official C++ or Python Software Development Kits (SDKs). It allows for high-level, module-based programming accessing features like motion and navigation, audio, vision, people perception, sensor access, trackers as well as diagnostic tools. The only major difference between using Python and C++ is that programs written in Python can be run directly on a computer or the robot, while C++ needs to compile the code for each specific platform. This is because C++ is a compiled programming language, while Python is an interpreted one.

The Robot Operating System (ROS)

An alternative way is to use the open source Robot Operating System (ROS) [27]. Its goal is to simplify the process of making complex and robust robotics software that works across different robots and platforms. In other words, they want developers to collaborate and build upon each other's work, making one able to use and modify systems made of experts in different fields for different robot platforms. It is designed to be distributed and modular, making one able to use as much or as little as one likes for individual projects. There exist a lot of user-contributed packages that can be used on top of the core system, and ROS has e.g. a driver for the newest version of the NAOqi Framework, which makes it easy to use ROS for development on the Nao platform [26]. It also includes a complete model of the robot, making the process of embedding it in simulators such as Gazebo or others easier.

3.2 Implementation of an integrated evolutionary learning platform

The platform should be able to test out and implement different EAs, as well as make it easy to access the robot's controller system and simulator.

Having this the ability to conduct tests and other experiments using the same program is also desirable.

3.2.1 Choosing a capable learning module for parameter tuning

As has been described in section 2.1.2, Genetic Algorithms (GAs) are one of the most popular learning methods in evolutionary computing. Since it has been utilised with success in many experiments before, such as in the relevant paper described in section 2.5.2, this kind of method was decided to be suitable for parameter tuning in this project. Another reason to choose this type of approach is that due to its popularity, it is easy to find complete implementations for programming languages, taking away the need to start writing code from scratch when testing out different algorithms. Since GAs are highly modifiable when it comes to features like chromosome representation, genetic operators such as crossover and mutation as well as the concept of elitist selection, finding an implementation of a simple GA and integrate it into the learning platform was decided to be the first step. After finishing this, setting up some preliminary experiments will hopefully lead the process towards more interesting challenges and algorithms, such as e.g. dealing with the reality gap using the Transferability approach described in section 2.2.2. This method utilises the NSGA-II algorithm, and the reason for choosing it is described in more detail in section 3.3.

3.2.2 A setup capable of evaluating solutions in both simulator and reality

As has been stated in the previous chapter, a desirable feature of the learning platform is to make it capable of testing out and evaluating solutions both in the chosen simulator, as well as on the real robot. In other words, it should be able to connect to and control both a simulated version and a real version of the Nao without large modifications to the setup. The way this has been implemented can be seen in the right part of Figure 3.5, where the control system, i.e. the MySupervisor class controls a Move-class that controls the robot using the NAOqi Framework. This is done independently of whether the robot is in a simulated or real environment, since the NAOqi OS that is implemented in the robot hardware, also runs on the simulated version. By knowing the IP address of the robot, one can therefore connect to and control the robot in the same way for both environments using modules found in the API. These modules uses different proxies and so-called brokers for controlling the robot, e.g. changing the robot posture using a Posture Proxy.

Because of the reasons mentioned in section 2.2.1, only experiments testing out solutions already evolved in simulation has been conducted on the real robot, although the implementation also allows the evolutionary

optimisation to be run directly on the physical robot. This is shown in the left part of Figure 3.1, where the evolutionary learning process in simulation, i.e. the parameter tuning is shown in red, and the testing is shown in blue. While the Move object representing the robot control system also could test solutions on the real robot (blue line), it was decided to make this part separate of the rest of the system, due to it only requiring testing the already evolved solutions, and not do any evolution itself.

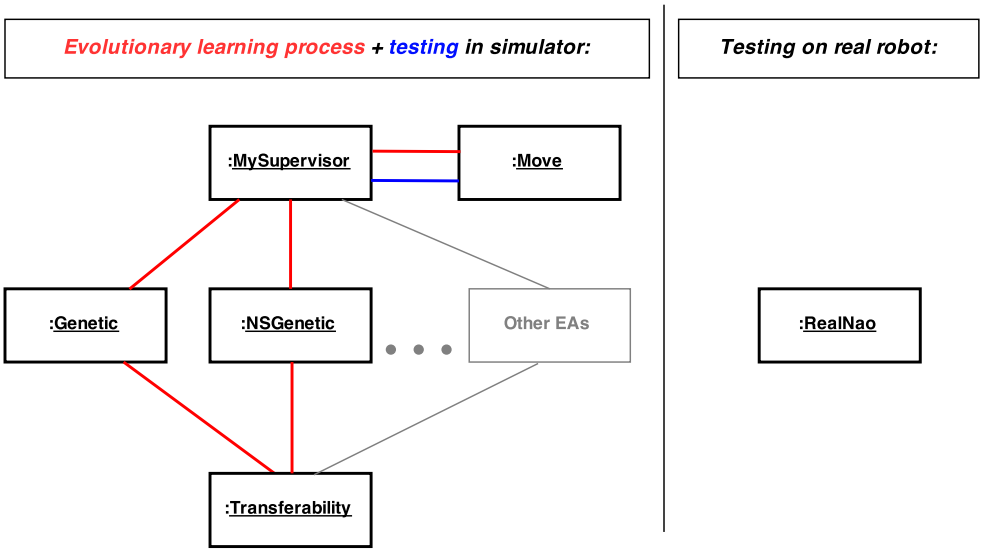


Figure 3.1: Left: A Unified Model Language (UML) object diagram showing the control system's relevant objects when doing evolutionary learning (red) and testing (blue) in the simulator, independent of the Evolutionary Algorithm used. Right: UML showing the relevant objects created when testing on the physical Nao robot.

3.2.3 A suitable control system for the experiments

The RealNao module found in Figure 3.1 controls the physical robot using the NAOqi Framework the same way as for the mentioned Move-object. The only difference is that the program conducting the tests on the physical robot has been separated from the rest of the implementation, due to it not needing any EA modules. The module handling all the requests involving the simulator, is the MySupervisor. Examples of requests can be to restart the simulation, as well as recording the three-dimensional movement of the robot, as well as resetting it to a start position between the runs. The whole process of accessing these external libraries are more described in section 3.4.

3.2.4 A suitable simulator

For simulating and evolving the gait parameters of the robot it was decided to use the previously mentioned Webots[78] simulator. There are several reasons for this choice: First of all, it has official support for the Nao robot, and one gets access to a free, Nao-specific version of the simulator if buying the robot. A second reason is that due to this official support, accessing and controlling the robot in the simulator works exactly the same way as for the real one. This is because the simulated robot runs the same operative system as the physical one, and therefore you can use the official API to access it. A disadvantage of this is that you cannot speed up the simulation to run faster than real-time, but this was found to not be a problem for the described experiments. The reason for this is found in section 3.5 on page 50.

Webots has the possibility of speeding up the simulations using two modes where the first one shows the faster simulation in the same three-dimensional view as when running real-time, and the second one where the simulation is run as fast as possible, i.e. as fast as the computer is capable of. This view however does not show any graphical view of the simulation. Utilising these two modes should be possible with the Nao as well, and is interesting for use in future works. To make this work, one has to modify the joint positions of the Nao robot node directly using the built-in Webots API instead of using the Nao API. This way of doing it effectively removes the possibility of using the API's optimised walking methods depicted in 2.4.4, and forces one to implement its own walking method, increasing the difficulty and workload of the task. A disadvantage is therefore that one can not utilise all the built-in stability features that are found in the official walking method and needs to take care of this oneself. When one has evolved or optimised a satisfying walk in the simulator, the transfer onto a physical Nao should on the other hand not be too difficult. This is because accessing and modifying the joint positions on the real Nao directly using the Nao API should be similar to the way it is done in Webots. Another possibility that has not been explored further is to simulate several robots simultaneously. When a simulation starts, each robot controller is started as a separate process, which is in turn connected to its robot node. Therefore, it should be possible to run several Nao nodes and access them using both the approaches described here at once, making it possible to do more evaluations in less time. This will naturally require some modifications to the controller system, e.g. using two server robot nodes will require two distinct access ports. Making the same controller system capable of accessing both the simulated and the real robot should make up for the lost time running the simulations in real-time.

Another reason for using Webots is that it has been utilised in similar experiments with the Nao in the past, like in the previously described papers [75] and [47]. A Webots simulation in action can be seen in Figure 3.2 on page 44, showing an evolution of a fast walk experiment using the

Transferability Approach. A final reason worth mentioning is that the control and supervisor files used in the simulator can be written in a variety of programming languages, including C++. This has made it easy to write and organise all the modules such as control systems for the simulated and physical robots as well as programs used in the simulations in the same Integrated Development Environment (IDE).

3.2.5 A suitable programming language

All of the programs used to communicate with both the physical and the simulated Nao have been written in C++. The program used to control features in the simulator was also written using the same language, although some small parts were written in C.

C++

C++ is a compiled, high-level programming language evolved from C, and was originally called "C with classes"[71]. This is because it supports high-level features like object-orientation, e.g. classes, as well as generic features using templates, in addition to low-level memory manipulation using standard C syntax. It is currently the third most popular programming language according to TIOBE Software[9], and is designed with performance, efficiency and flexibility in mind. Several platforms offer different implementations of the language, like Visual C++ using Microsoft Visual Studio and the open Gnu Compiler Collection (GCC) for Unix. The NAOqi Framework has a Software Development Kit that comes in both Python and C++ versions, although other languages are supported for communication with the robot as well. Since the Webots simulator also has a C++ API, it was therefore decided to use this language for the whole project, using different modules for communicating with the robot and simulator, as shown in Figure 3.5.

Microsoft Visual Studio

Visual Studio is an Integrated Development Environment (IDE) made by Microsoft, and it supports making computer programs for Windows with several different programming languages, such as C++ using Visual C++, and `C#` using Visual `C#`. It is the preferred IDE to use with the Nao API on the Windows platform, in addition to being able to make Webots simulator controllers, and was therefore chosen as development platform for the project.

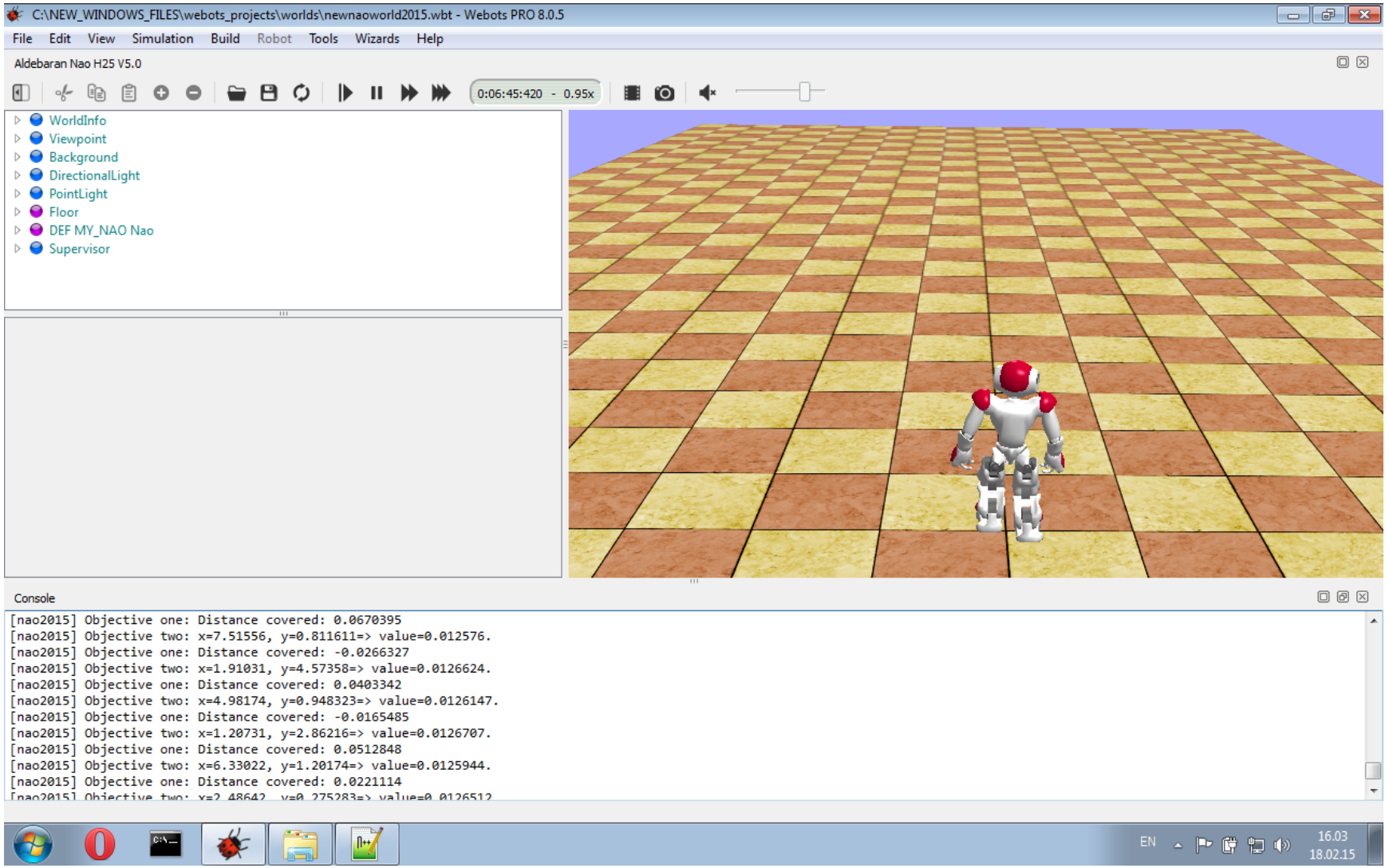


Figure 3.2: Evolution of walking gaits for the Nao robot in the Webots simulation software. The MOEA NSGA-II tries to find a fast walking gait(objective one) that transfers well to the real robot (objective two) using the Transferability Approach.

3.3 Evolutionary Algorithms for the experiments

As described in chapter 2, there are several kinds of Evolutionary Algorithms one can use to evolve for example walking gait parameters. To test out the robot platform, a reasonable idea would be to just first implement a simple Genetic Algorithm, and observe if one gets the expected outcomes or not. By doing this, one will get a good first impression about what ideas and problems it will be interesting to further explore in later experiments. Choosing the appropriate algorithms and techniques for these experiments will naturally also be a part of this. A preliminary investigation consisting of implementing and using a simple GA was therefore decided to be done. For the rest of the experiments, the algorithm called NSGA-II described in Chapter 2.1.4 on page 9 was decided to be implemented. The reasons for choosing this EA can be found in the next section.

3.3.1 NSGA-II

There are several good reasons for choosing the NSGA-II for evolutionary optimisation and EMO problems. First of all, as has been mentioned; it is a very popular algorithm. This means that it is easy to find an implementation in one's preferred language, removing the need for implementing it from scratch. The modified version that has been utilised in this project is implemented as shown in Figure 3.6, using the functions described in the Evolutionary interface for setup and control. It's well-documented behaviour has been utilised in similar projects and experiments before with success, where an example can be the multi-objective optimisation dealing with the Reality Gap problem using the Transferability Approach. It can also be used for problems having only one objective, and it's sorting algorithm is known to be very effective. As it also incorporates elitism or elitist selection when optimising one objective, it can be interesting to compare it's results with other Evolutionary Algorithms that have other selection operators. The genetic operators implemented in this project are described in the following section. Since the newer version of the algorithm, NSGA-III is concentrated on problems having more than four objectives, this algorithm has not been considered used for the experiments described in this thesis. This is because the maximum number of objectives for the optimisation problems conducted here are two (Transferability Approach).

3.3.2 Genetic operators

Since the genotypes are real-valued, one can use the same crossover operators for recombination as for binary represented ones. This has been done for the simple GA, which uses a discrete, single-point crossover, while the NSGA-II uses a different one called Simulated Binary Crossover

(SBX). This operator decides in general how individuals from the parent generation are combined to form new individuals for the succeeding generation, i.e. modifying the parents' individual chromosome structures. The so-called crossover rate or probability decides how many of the selected couples that will be undergone crossover, i.e. mating. The others will survive into the next generation either unaltered, or will undergo mutation. How the parent selection works is described in one of the next paragraphs. In all experiments in this thesis the crossover ratio has been set to 1.0, recombining all the selected couples.

When it comes to the mutation operator, which goal is to maintain genetic diversity, the simple GA uses a uniform mutation technique, where the elements can be replaced by a number between the limits specified by the genotype representation. It is decided by the mutation rate, and this probability decides how likely it is that random elements in a chromosome get changed into something else, e.g. a bit flip if using a binary genotype representation. The NSGA-II uses another mutation called polynomial mutation. All the experimental setup details can be found in Chapter 4.4 on page 66.

Before conducting these two operators, a part of the current population needs to be selected as a basis for the new one. It is therefore worth noting that the couples that are used for recombination, i.e. crossover, as well as mutation are not picked directly from the current population, but have undergone the last genetic operator called selection. There are several ways to decide which parents to select, but in general it starts by ranking them according to their fitness's. Maybe the most common one is called fitness proportionate or roulette-wheel selection, and works by randomly selecting individuals for recombination until one reaches the desired number. The fitness values are however in this case used as a probability to get selected. By doing it this way, the risk of selecting only poor individuals is apparent, but since the probability of getting chosen for the best individuals is much higher, this undesirable possibility is reduced. It can easily be pictured by throwing a ball on a roulette wheel, where the individuals with the highest fitness have larger slots for the ball to fall into, and therefore a higher chance of getting selected.

The genetic operators used in this project are all described in the following sections.

Single-point crossover

The single-point crossover operator is depicted in Figure 3.3. The crossover point is selected at random, splitting the two parents at this point. The two resulting children is made up of one part from each of the parents.

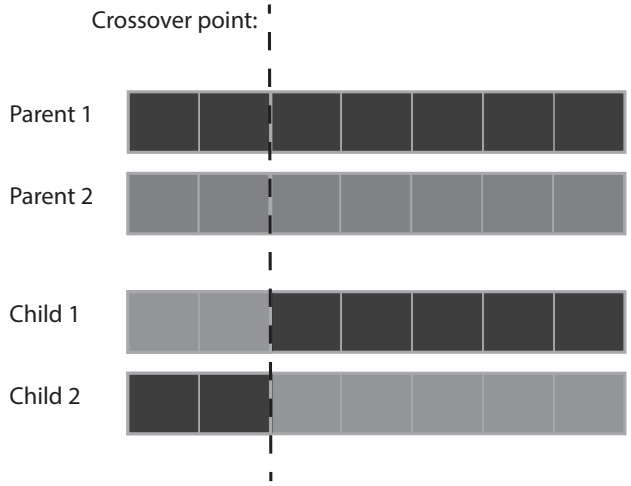


Figure 3.3: The Single-point crossover GA operator

Uniform mutation

The uniform mutation operator is shown in Figure 3.4. As one can see, a random gene in the unmodified genotype is selected to be mutated. This gene gets replaced by a uniform random floating point number chosen from the specified ranges, e.g. like the maximum and minimum values found in Table 3.1 on page 54.

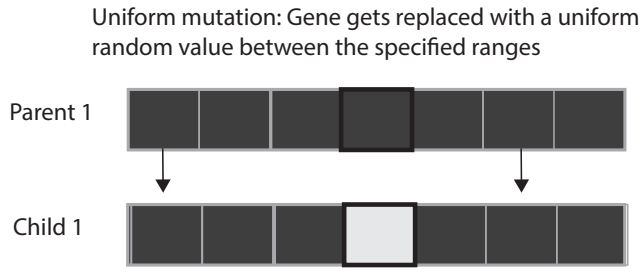


Figure 3.4: The Uniform mutation GA operator

Rank-based selection

Unlike the previously described roulette-based selection method, the rank-based sorts the individuals by giving them a score, rather than using the fitness directly. The result of this is that the best individual is only a rank better than the second best one, e.g. giving them a score of 1 and 2,

respectively. In the other approach on the other hand, the best individual would have a ten times higher score than the second best one if the fitness was ten times higher. The rest of the process works the same way, i.e. using these weights for the random selection.

Simulated Binary Crossover (SBX)

The NSGA-II uses an operator called the Simulated Binary Crossover (SBX)[17], and is inspired by the single-point crossover operator used in binary-coded GAs, similar to the one described earlier. It differs however by trying to simulate it to have the same search power by using a so-called density function. An important property of the binary crossover is that all the children's gene values have the same distances from the parents' average gene value. In other words, the average values of the decoded parameters are the same both before and after the crossover operation. Another is that if one takes crossover in one of the lower bit positions, the resulting gene value change will be small. Children are also more likely to be close to their parents than for other chromosomes. These are the reasons why the density function is needed, instead of just using the single-point crossover technique directly. The algorithm is described below:

- Two parents *x1* and *x2* are selected as normal.
- Then a random number u is generated between 0 and 1.
- Then a spread factor $\beta$is calculated using the density function found in Equation 3.1 and the distribution index ηc.
- The offspring are then computed using Equation 3.2.

$$\beta = \begin{cases} (2u)^{\frac{1}{\eta c+1}}, & \text{if } u \leq 0.5 \\ (\frac{1}{2(1-u)})^{\frac{1}{\eta c+1}}, & \text{otherwise} \end{cases} \tag{3.1}$$

$$x_1^{new} = 0.5[(1+\beta)x1 + (1-\beta)x2] \tag{3.2a}$$

$$x_2^{new} = 0.5[(1-\beta)x1 + (1+\beta)x2] \tag{3.2b}$$

It is worth noting that

- $(x_1^{new} + x_2^{new})/2 = (x_1 + x_2)/2$

holds for this simulated operator, showing that one still have achieved the same as the ordinary binary version, i.e. that the two children's gene values have the same distances from the average gene value of the parents. When it comes to the distribution index, a large one is supposed to generate children closely related to their parents, while a smaller one allows for children located far from their predecessors. This crossover has been shown to have similar or better results compared to the binary version, especially in problems having multiple optimal solutions [17].

Polynomial mutation

The NSGA-II algorithm uses a polynomial mutation operator, and is calculated using:

$$c_k = p_k + (p_k^u - p_k^l)\delta_k \tag{3.3}$$

where child number k, c_k is given by the parent p_k using the lower and upper limits on the specific gene, p_k^u and p_k^l, respectively. The δ_k is used as a little modification which is calculated from a polynomial distribution, hence the name of the operator.

Binary tournament w/ crowded comparison operator selection

An explanation of this selection operator can be found in the last paragraph in chapter 2.1.4 on page 9.

3.3.3 Elitism

Like mentioned in the NSGA-II sections, elitism or elitist selection ensures that the best solutions in the current generation survives in an unaltered way into succeeding generations. This is to ensure that the overall quality of the solutions are not degraded during the optimisation process. The elitism rate decides how many of the individuals in the current population that survives into the next generation. If this e.g. is set to 1.0, all the individuals survives into the next generation, if not any better solutions are found. With an elitism rate of 0.1 on the other hand, only 10 percent of the current population can potentially survive into the next generation. Another way to decide the elitism impact can be to decide a number of surviving solutions, but this has not been used in this project.

3.4 A learning module with the capability of testing various EAs

The module-based control system shown in Figure 3.1 and 3.5 is made bearing in mind the ability to test out various Evolutionary Algorithms such as a simple GA and the NSGA-II in an easy way. The way this is done is to require modules of different Evolutionary Algorithms to implement an interface called *Evolutionary*, requiring two methods called init and start. By doing it this way, all the evolutionary optimisation is done in this separate module, using pointers to the modules described in Figure 3.5 for accessing and controlling the simulator and robot. This Evolutionary module is also responsible for outputting and writing to file the results obtained. In Figure 3.1, one can see how implementing

another Evolutionary Algorithm module would fit in to the already existing learning platform and process. A more specific example can be seen in Figure 3.6, where an implementation of the multi-objective NSGA-II algorithm is initialised and started by the MySupervisor module using the methods described in the *Evolutionary* interface. This specific implementation utilises a package called 'nsga2', that is not required by other EAs. It also utilises the optional 'Transferability Approach' module when optimising two objectives concerning the reality gap. This is described in more detail in Section 4.2.2.

3.5 Bipedal locomotion control using the NAOqi Framework

The NAOqi Framework makes it possible to achieve locomotion control using three different high-level functions. The function used in this thesis, is called move(), and uses a so-called motion proxy to communicate with the robot. To set up these proxies, one first needs to establish a connection to the NAOqi OS server using its IP address and an open port, which is running on the physical robot or as a standalone process on a computer, either in a terminal window or in the Webots simulator. The function makes one able to set the robot's immediate velocity in x- and y-direction, as well as its rotation around the z-axis. In addition to this, one can alter the walk by defining so-called custom gait parameters. This allows for customising the walk while still using the ordinary motion API, as an alternative to e.g. controlling the positions of the joints directly. The API also offers functionality to do this by controlling individual or several joints at the same time using a high- level device control manager (DCM) with interpolation to make for a smoother behaviour. This method should be similar to a general low-level way of accessing and controlling the robot and it's joints.

The first mentioned approach has been found sufficient for the experiments involving controlling and evolving the walking gaits in this thesis. A reason for this is because the thesis is more about investigating the opportunities of the Nao for evolution, rather than to discover a superior walking gait compared to the ones found in previously mentioned research papers, such as in [47] and [64]. This has made it easy to focus more on exploring concepts like the Reality Gap, as well as making the module-based evolutionary learning platform for the robot with the capabilities of e.g. easily testing out different EAs. There are also several advantages of using the NAOqi Framework found in the last paragraph of this chapter, and is another reason for this choice. It should also be possible to later expand the functionality of the learning module by implementing a more low-level controller that accesses the joints directly e.g. by using the high-level DCM with little effort. This will make the setup capable of testing out other kinds of optimisation problems, such as making the robot able to

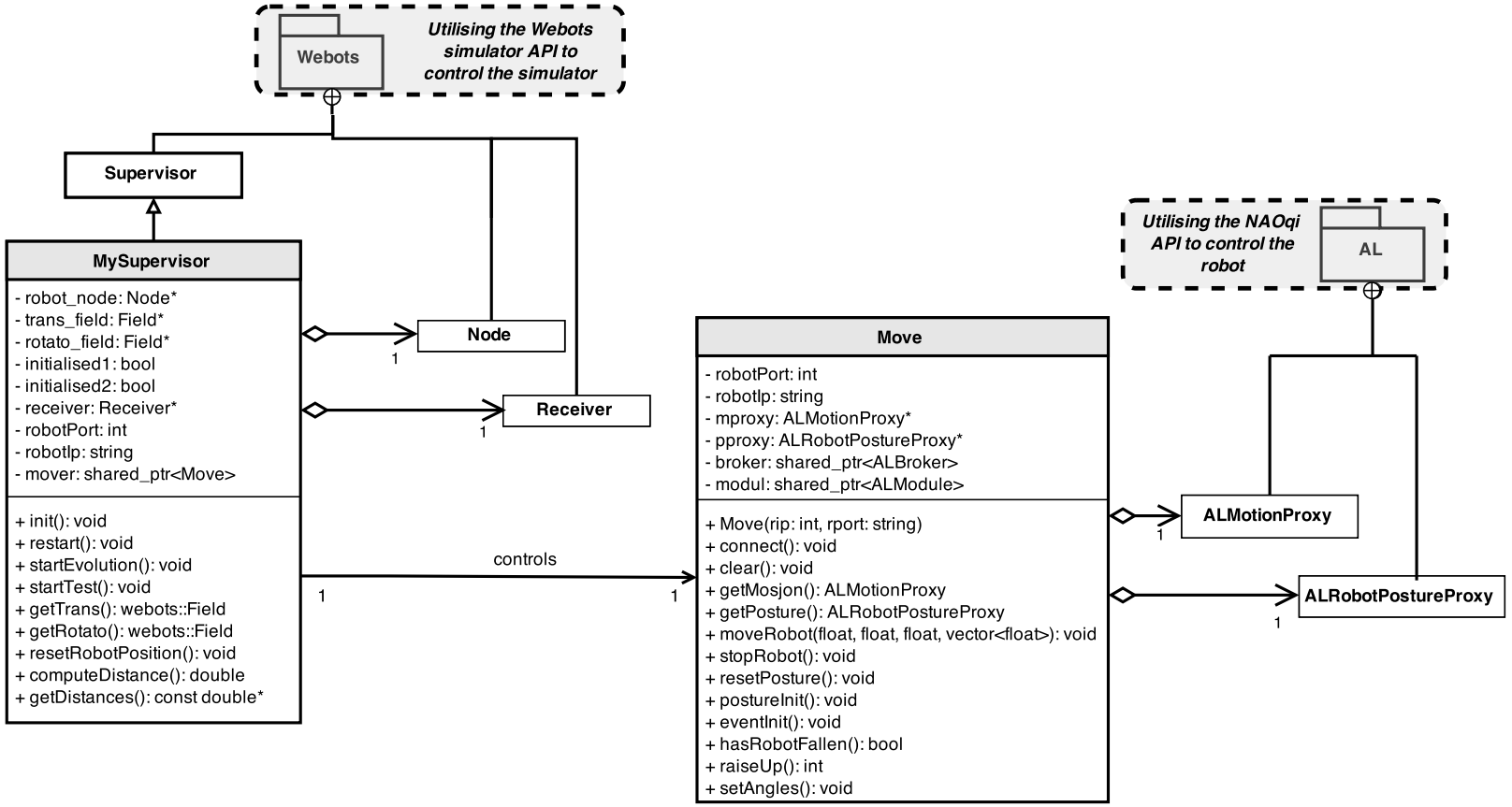


Figure 3.5: A Unified Model Language (UML) class diagram showing the control system's use of the external APIs. The Webots simulator API is used to control everything regarding the simulator and simulation, while the Nao's API called the NAOqi Framework is used for communication with the robot.

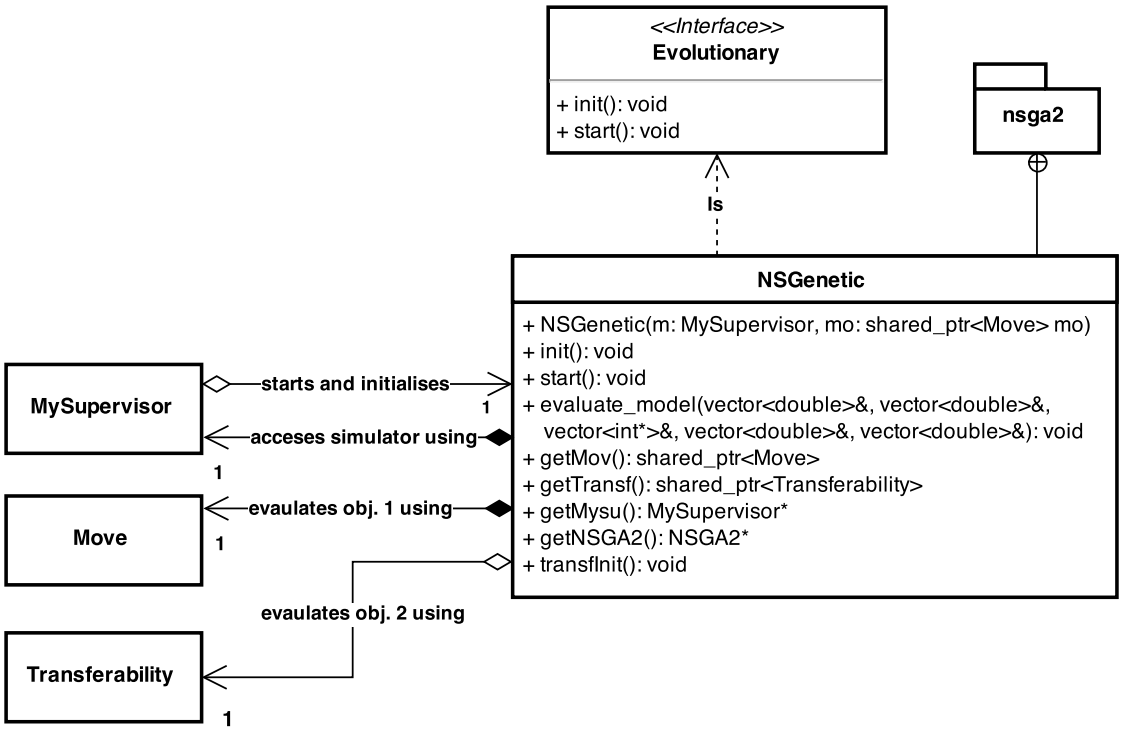


Figure 3.6: A UML class diagram showing the implementation of the evolutionary learning module NSGA-II. The system is module-based, making one able to do evolution using different Evolutionary Algorithms, as long as they implement the *Evolutionary* interface.

crawl.

In Table 3.1, one can see the different parameters that the user can set to alter the gait of the robot. For the purpose of this thesis, modifying gaits using these parameters to find an optimal walk using an evolutionary approach can be an interesting task. As one can see, the different parameters have different ranges, but there is not really any guidelines for setting good or bad parameters except the listed default parameters. These are however tuned to make for a more stable and efficient walk, rather than a fast one. The only suggestion is that increasing the parameter called 'MaxStepX' to a value of more than 0.060 can possibly make the gait unstable if not using the robot on a flat, hard surface. To make this a constraint for the evolution has not been taken into account, as it can be interesting to see if the robot can find stable walks also using parameter values above this threshold, or in fact discover this limit on its own. Another thing worth mentioning is that the parameters can be set per foot or for both feet. If one specifies the parameters for only one of the feet, the other one will use the default parameters.

Compared to using a low-level controller for locomotion, the NAOqi Framework comes with several advantages, in addition to making it easier

to use. First of all, when the robot walks; the system is constantly monitoring and utilising the feedback from its joint sensors, trying to stabilise the walk. The purpose is to make it more robust when it comes to dealing with small interferences by absorbing potential torso oscillations in the frontal and lateral planes. The frontal plane is easily visualised by standing with your back against a wall. Arm, leg and back motions that can be conducted while still touching the wall with your back are so-called frontal plane movements, and divides the body into back and front portions. The lateral plane on the other hand divides the body into left and right portions. According to the official documentation[58], the Nao is able to walk on and transfer between hard surfaces such as floors covered by carpet, or tiles. However, it also states that the robot assumes a more or less flat surface without any major obstacles, due to the ZMP-based walking method explained in chapter 2.4.

Parameter name	Function	Default value	Minimum value	Maximum value
X	Velocity along the X-axis (m/s)	N/A	N/A	N/A
Y	Velocity along the Y-axis (m/s)	N/A	N/A	N/A
Theta	Velocity around the Z-axis (rad/s)	N/A	N/A	N/A
MaxStepX	Maximum movement along the X-axis, i.e. forwards (metres)	0.040	0.001	0.080
MaxStepY	Maximum movement along the Y-axis, i.e. sideways (metres)	0.140	0.101	0.160
MaxStepTheta	Maximum rotation around the Z-axis (metres)	0.349	0.001	0.524
MaxStepFrequency	Maximum normalised step frequency (no unit)	1.000	0.000	1.000
StepHeight	Maximum elevation along the Z-axis (metres)	0.020	0.005	0.040
TorsoWx	Maximum torso rotation around the X-axis (radians)	0.000	-0.122	0.122
TorsoWy	Maximum torso rotation around the Y-axis (radians)	0.000	-0.122	0.122

Table 3.1: A table showing all the gait parameters that can be altered or optimised using evolutionary optimisation plus the three velocity parameters in the top.

Chapter 4

Experiments

Although the Nao platform seemingly is capable of a lot of different kinds of Evolutionary Robotics experiments, it was decided in this project to mainly focus on exploring the possibilities of optimising walking gaits, i.e. locomotion learning, for various challenges. Since the implementation has been made in a way that makes it able to use different Evolutionary Algorithms, testing out and exploring the performance of different EA implementations is naturally of great interest in this kind of project. This not only to explore the possibilities of the Nao platform itself, but also later on to find out how the actual implemented learning platform performs bearing in mind both its capabilities and performance.

All the optimisation and testing has been done using different modules in the same program implementation, as explained in the previous chapter and in Figure 3.1.

4.1 Purpose of the experiments

The main purpose of the thesis is to investigate the Nao platform's suitability for doing Evolutionary Robotics. One part of this study is therefore to actually do some evolutionary experiments, such as experimenting with different Evolutionary Algorithms as well as looking into challenges such as the previously described Reality Gap problem. The conducted experiments can roughly be divided into three parts:

1. An evolutionary optimisation part in simulator.
2. To transfer and test out the solutions on the real Nao robot.
3. A comparison part involving the optimised solutions' behaviours in simulation and reality.

A detailed explanation of these parts follows in the next couple of sections, including a description on the necessary integrated data collections. The results and analysis of them can be found in the next chapter.

4.2 Suitable optimisation problems for the Nao

There are several different kinds of interesting optimisation problems when it comes to humanoid robots. The experiments chosen for investigating the platform's suitability for Evolutionary Learning in this paper were decided to be divided into three parts, each with its own purpose, but all concentrated on bipedal locomotion learning:

- A preliminary investigation setting up and conducting a walking gait optimisation on a flat surface using a simple Genetic Algorithm.
- Investigate the Reality Gap for this kind of problem, and test out the necessary efforts to deal with it (using the Transferability Approach).
- Look into the possibilities of walking gait optimisation on a more uneven surface, and how it affects the stability and performance of the robot.

The experiments are more described in the following sections:

4.2.1 Bipedal locomotion learning

The common, classic problem of walking gait optimisation seems like an interesting focus field for investigating ER experiments on the Nao platform. Particularly the optimisation of bipedal locomotion gaits is of interest, since the Nao is a humanoid robot with two legs. As has been mentioned in chapter 2.4.1, there are several different factors to consider when finding good walking gaits. Since the walking method found in the Nao API is mainly optimised with stability in mind, it was decided to use this as a basis for optimising faster walking gaits, while still maintaining stability.

For the experiments, the fitness function was decided to be the distance covered by the robot during the evaluation period in the x-direction, i.e. forwards, with the subtraction of the absolute value of the movement in y-direction, i.e. sideways:

$$f_{Fitness} = Movement_x - |Movement_y| \tag{4.1}$$

By subtracting the distance walked in the y-direction, the goal was to make the robot walk as straight ahead as possible, by favouring long walks with as little sideways movement as possible. It was also decided to give a negative value of minus one to gaits making the robot fall over, effectively making these individuals' fitness values worse than all others. Implementation-wise, the fitness value used for optimisation was actually set to be 10 subtracted by equation 4.1, as the implemented NSGA-II actually tries to minimise the objective. As the fitness value is measured in metres, the value will most likely never be more than 10 for a 10 seconds long evaluation period, making the best fitness values the smallest. As for the penalty of -1 for gaits making the robot fall, the value will now become

11, even further away from zero, effectively making it worse than all other fitness values.

The evaluation period was set to 10 seconds, as this was found to be a long enough time to find good gaits. A problem that can occur when using a shorter evaluation time is that some gaits seem stable at first, but after some time they get unstable due to oscillating effects, making the robot fall over. An example of this can be that the gait makes the robot start to swing from side to side while walking. This swinging can start to escalate, creating an oscillating effect, and it was observed during experiments that this effect could make the robot fall over after passing the evaluation time, making it not having any impact on collected the fitness value.

Because the gait parameters for the walking method shown in Table 3.1 on page 54 are real-valued numbers, it was decided to use the same real-valued representation for the genotype in the Genetic Algorithm. Each element in the genotype is for that reason coded as a floating point value, and restricted to the ranges decided by the minimum and maximum parameter values listed in the table. Effectively, the genotype is therefore in this optimisation problem represented by a vector containing the seven walking gait parameters.

4.2.2 Transferability Approach to deal with the Reality Gap

As mentioned in the background chapter, the Reality Gap can be a significant issue when it comes to transferring solutions optimised in a simulator to reality. After conducting some preliminary evolutionary optimisation experiments, and transferred some of the best solutions to reality, it was discovered a Reality Gap between the two environments. See sections 5.3 and 5.4 for more about this. The most observable gap was discovered to be that the robot during the evaluation period walked further and further out to the left, although this did not happen in the simulator. In fact as mentioned in the previous section, the optimisation problem's fitness function was made to take sideways walking into particular consideration, preferring straight walks over omnidirectional ones. The previously described Transferability approach was therefore decided to be utilised and implemented, in order to test out and observe if this Reality Gap could be minimised in any way. Another side-effect of this was to investigate the possibilities of using an Evolutionary Multi-objective Optimisation (EMO) on the Nao platform.

A point worth mentioning is that the approach requires the simulator to be accurate in some sub-parts of the search space in order to make the mapping work, and has therefore been assumed when using this approach. As the Reality Gap was less observable when considering other aspects of the walk, e.g. that the stable walks in the simulator never fell over when transferred over, this is a reasonable assumption.

Project specific

When it comes to the transferability mapping in this project, the mapping from a genotype space is infeasible, due to it consisting of as many as seven genes, i.e. parameters. This would require a transferability function of eight dimensions, i.e. the seven genes and a transferability score, and was therefore not considered:

$$Transferability_{score} = Transferability_{func}(p_1, p_2, ..., p7) \tag{4.2}$$

, where the transferability score is the difference in fitness values between simulation and reality, and p1 to p7 are the seven gait parameters in the genotype describing the solutions.

As explained in the previous section and shown in Figure 2.4, the approach of mapping the transferability score using a typical behaviour in the simulator was chosen instead. To test out this approach, a simple behaviour consisting of the omnidirectional movement in the simulator was chosen to be used for the mapping. By doing it this way, the function would work like this:

$$Transferability_{score} = Transferability_{func}(b_1, b_2) \tag{4.3}$$

, where the transferability score is the difference in fitness values between simulation and reality, and b1 and b2 are the two chosen behaviours in the simulator that are supposed to describe the solutions.

More specifically, the first behaviour used as an input was the distance covered in the x-direction during the run. The second behaviour used was the distance travelled in the y-direction. When it comes to the similarity measure, the difference in fitness values was used. The fitness was set to be the covered walked distance over the evaluation period, as used in the previous experiment, and listed in equation 4.1. This fitness function is therefore the first objective, while the transferability function is the second one. However, it was discovered that these two behaviours are effectively the same as the ones that are used to calculate the fitness values. This is probably not desirable, and can make one question the usefulness of the particular function. A better function will probably be to use another behaviour, such as the ones mentioned earlier.

It was therefore decided to make another transferability function using the average height captured at timed intervals during the evaluation period as an input space. The similarity measure was decided to be the same as for the first one, i.e. using the fitness function consisting of the covered forward distance subtracted by any sideways movement. After exploring this way of making and mapping the transferability function, it was decided to not continue with the experiment. The major reason for this was the discovery that the average height did not differ in any significant way for the distinct walks, making it rather useless as a solution descriptor. This is probably

due to the way the walking method works, described in Chapter 2.4.4 starting on page 25.

Since the only real reality gap was the slipping making the robot walk more and more to the left, it was decided that experiments exploring other factors of the walk, e.g. the stability on uneven terrain, was more interesting to conduct. The remainder of the experiments therefore concentrates on those matters.

4.2.3 Locomotion learning on uneven terrain for stabilisation

According to the Nao documentation, the walking methods supplied by the NAOqi API assumes the ground to be more or less flat and without any major obstacles. This makes it interesting to explore the possibilities of evolving custom gait parameters to make the robot able to walk on a more uneven surface, such as uphill and downhill or to deal with small obstacles such as tiny boxes or cables. It is also interesting to see how this will affect the gait stability. Observing and comparing gait parameters such as *StepHeight* in Table 3.1 on page 54 on solutions from this approach to when evolved on a flat surface can be interesting. It is for example reasonable to assume that taking too long steps, i.e. a high step length while walking uphill can make the robot fall over. The step height is also perhaps higher than when evolving gaits on a flat surface, to make the humanoid able to follow the track uphill, or to make it able to walk over small obstacles. This will naturally depend on the specific terrain, and is described in the next section.

The terrain used for optimisation

The terrain used for walking gait optimisation on uneven terrain in the simulator can be seen in Figure 4.1 on the next page. It pictures the Nao robot in its start position on a flat floor, facing towards a ramp that goes up and on from this surface. During the experiment phase, it was tested out different gradients for these ramps, as making the angle too small made the robot able to proceed over the terrain without any optimisation. A too steep ramp inclination on the other hand would naturally make it impossible for the robot to walk over, making it an infeasible task even for an evolutionary optimisation. After some experimentation, the gradient on the ramps were set to an angle of 0.1745 radians or about 10 degrees. This was found out to be a not impossible task for the robot to achieve, while still making it a challenge. This because using the default parameters would make the robot fall on its back when proceeding onto the ramp.

It seems like there are three major challenges one needs to overcome when walking on this kind of terrain. The first one is that the robot is more prone to slipping problems due to it having a harder time stabilising the body weight on the feet, meaning it needs to compensate for this by e.g. leaning

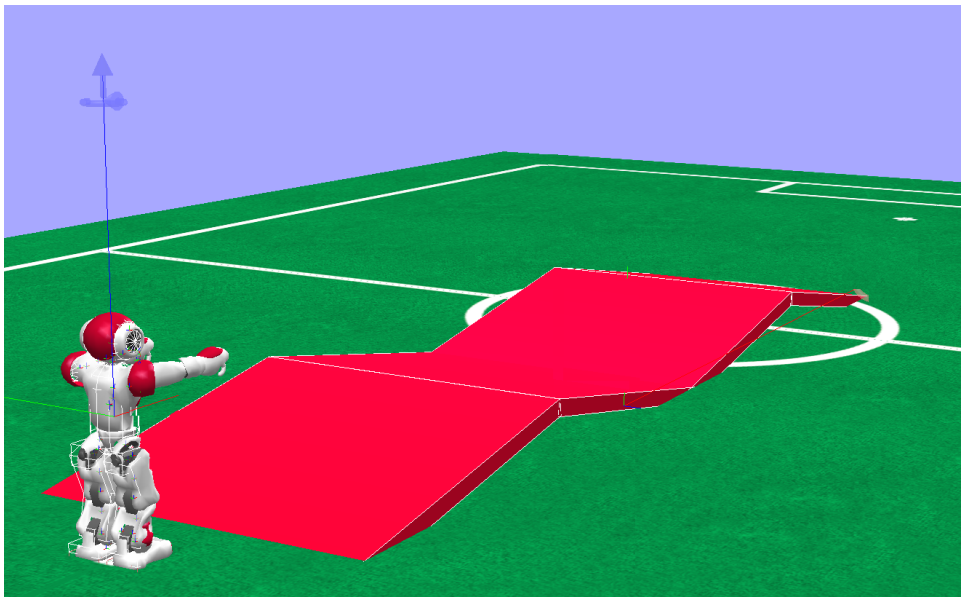


Figure 4.1: The uneven terrain used for optimising stable walking gaits in the Webots simulator.

the upper part of the body more forwards when walking uphill. The second and third problems arise from this, as the slipping can make the robot fall on its back, as well as making the gait more sideways. This can in turn make the robot walk and fall over the edges on the sides of the ramp. It was therefore decided to use the same fitness function as used in the previous gait optimisation experiments, previously described and listed in chapter 4.2.1 and Equation 4.1.

Other interesting terrains to use in the optimisation process can be e.g. on a flat surface to randomly place out and make obstacles such as small boxes or wires.

4.2.4 Factors possibly affecting performance

When conducting the experiments that deal with robot motion and movement, there are several things one needs to keep in mind. The already described choice of landscape and material is naturally important when making the robot walk, as well as avoiding any obstacles. Other factors can be that the robot's joint temperatures can vary during an experiment, possibly affecting the performance of late runs. This has been dealt with by not doing too many tests in a row, avoiding putting too much stress on the legs in particular. The battery life is also of concern and can make the performance worse, due to the legs receiving not enough power to move properly. The experiments have therefore been done with at least 25 percent of battery capacity left. The network cable has been attached at all times, and an initial test with and without this cable was conducted, showing no significant difference in performance. These two factors have

therefore been eliminated as sources of error.

4.3 Integrated data collection

In addition to finding interesting and suitable experiments for the Nao platform, having a focus on integrated data collection during the experiments is important. This has been done to make it easy to log data both during the learning loop, i.e. so one can observe what is going on, as well as making sure the results are saved for further analysis later on. Included in this collection is data from the optimisation and testing in the simulator, as well as motion capture data gathered from the real robot in the motion capturing lab afterwards.

4.3.1 During the learning loop in simulator

While doing the evolutionary optimisation experiments and conducting the tests in the simulator, collecting interesting data such as distance has been done using the built-in method that gives the three-dimensional robot position at all times. By subtracting the start position from the final position, the distance covered during the trials in both x- and y-direction is easily gathered. As for the Transferability Approach, measuring the average height of the robot has been done in the same way.

4.3.2 During the physical robot testing on laminated floor

To test out the performance and transferability of the evolved solutions found in simulation, the gaits were transferred onto the physical Nao robot. As a preliminary investigation, an experimental setup in the lab was made, with a fixed starting point, ruling out any differences due to the placement on the surface. The lab has a flat, laminated floor which can be quite slippery, and it will be interesting to see if this has any impact of the real performance, e.g. making the robot slip, creating a possible Reality Gap. A picture showing the laminated floor setup is found in Figure 4.2.

The actual testing was conducted using a measuring tape, measuring from the tip of the robot feet, as shown in the figure, to the same tip after the run. The line going through the middle of the robot was used as a baseline for any possible sideways movement, and was measured to the point on any of the feet the furthest away from this line. When it comes to the evaluation time, a stop signal was sent to the robot after passing the evaluation time, making the robot sit in a relaxed position like the one in the picture. This has been done in all the physical experiments, making comparisons between them easy.

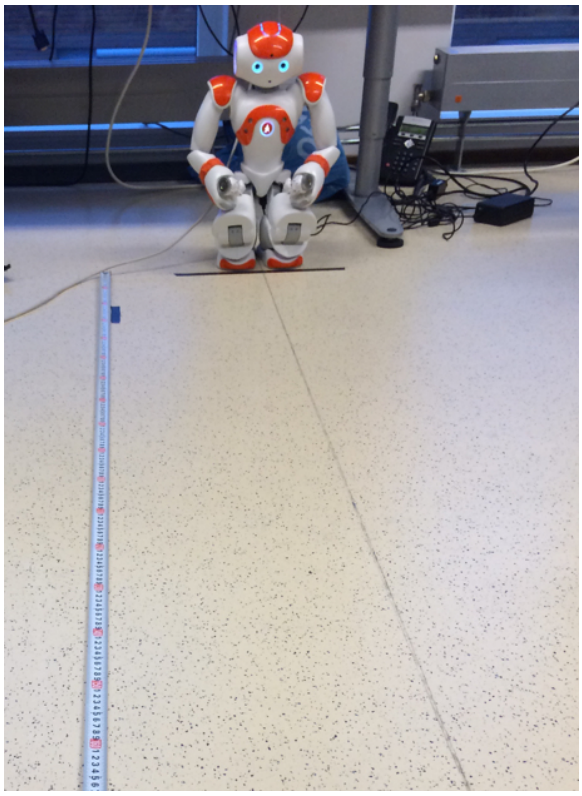


Figure 4.2: The Nao in the laminated floor test setup.

4.3.3 During testing in the Motion Capture lab

For more accurate measurements of e.g. robot movement, many in the professional and military industry as well as researchers use motion capture technology to collect these data. It is e.g. utilised in video games and movies to make the characters have a more natural movement than what is possible to create digitally. Motion capturing makes one able to measure both position and orientation of robots over time much in the same way as one can easily do in simulators. The motions are gathered and transformed to digital information that can be processed by computers. The way it works is that moving objects like robots are equipped with reflective markers that can be tracked by cameras. These markers are usually small balls which can be active or passive. Active markers have diodes that emits light, and they can be turned on individually when needed. Passive markers on the other hand do not have these diodes, but reflect more light than other normal materials. A problem that can occur with the passive ones is that the system can not differentiate between the different markers, degrading the tracking accuracy. Another problem with passive markers is that you need to cover up all the markers that are not in use but still visible for some of the cameras. The markers are captured using optical systems such as high-speed cameras to accurately generate their three-dimensional positions. Since the system is optical, it needs two-

dimensional images with a clear view of a marker from at least two distinct cameras to decide its position in three dimensions, just like when humans percept with their eyes. The Nao robot also uses two cameras to measure distances to e.g. possible obstacles. If a marker is not visible, its 3D position simply dissappears from the motion tracker.

Motion capture setup

The motion capture system installed at the research group is called OptiTrack. It consists of twelve high-speed OptiTrack FLEX:V100 cameras with integrated processing units, and captures up to 100 frames per second for accurate detections. One of the cameras is shown in Figure 4.3. Gray-scale images of 640x480 pixels are transferred via USB to a computer for further processing, and is combined with the ARENA motion capture software package, allowing for real-time rigid body tracking of both position and orientation. The cameras use infrared light to detect the motions of the markers on the robot, which are passive in this setup. All of this happens in real-time, making one able to watch the movements of the markers in the ARENA software while it actually happens. When it comes to working with and plotting the recorded data, the ARENA software can trajectorise the movements, i.e. make data points of the 3D points, which can in turn easily be exported to formats such as the binary coordinate 3D format called c3d. Using tools like the MoCap toolbox[8] developed at the University of Jyväskylä, one can import the points into tools such as Matlab with no hassle. This can in turn be used to visualise the data, as well as for analysis. An example of this can be to merge measurements from different runs and/or trackers of the same solutions together, making the results more robust.

During the experiments, the experienced accuracy was found to be very precise. An example showing this is that the height trajectories for the different markers while walking moves the same way, keeping the same height between them at all times. The observed slipping that made the robot walk further and further out to the left seemed to be tracked with a satisfying detail- level as well as the important forward movement distance. Compared to other methods, such as using measuring tape like was done on the laminated floor, this approach is superior in several ways in addition to the better accuracy: Since there is no need for measuring and noting down the results for each run manually, this approach is less time consuming, as the measurements are taken care of by the cameras. It is also easy to merge together several runs to make the results more robust.

Experimental setup

For the purpose of accurately testing the robot's movements, there has been placed 4 passive markers on the Nao torso, as this is the preferred

Figure 4.3: An OptiTrack FLEX:V100 high-speed infra-red Motion Capture camera used for real-time rigid body tracking.

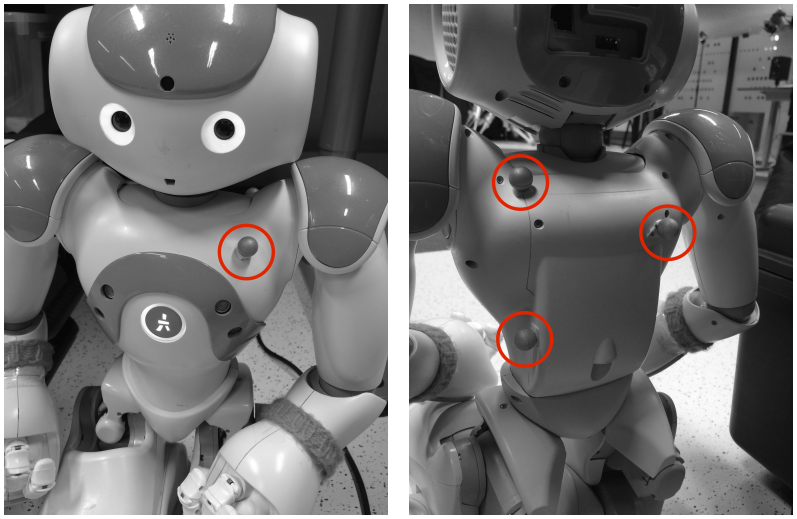


Figure 4.4: The placement of the passive markers on the Nao robot. The MoCap system recommends four markers placed asymetrically in relation to each other, and has as a consequence been utilised the same way in this setup.

number of markers according to the ARENA software. The reason to place all the markers on the torso, is because they should not move in relation with each other, which can confuse the tracking. The markers are placed asymetrically to avoid any possible ambiguous orientations, with one on the front, two on the back, and one on the side; trying to avoid it getting covered up by the robot arm. Other marker positions should be fine to use as well, since it was discovered that the biggest impact on accuracy came from the calibration of the cameras, and not from the placement of the markers. The cameras' setup can get uncalibrated only by a tiny displacement of one single camera, leading to poor or no tracking results at all. Therefore, the setup was always tested and verified before any tests were conducted. An example showing a typical behaviour of the Nao robot captured by the MoCap system can be found in Figure 4.5.

Because one is mainly interested in the walking gait of the robot, the motion capture equipment's ability to track movement in both x-and y-direction made it quick and easy to test out and measure different solutions, i.e. gait parameters. The ability to also measure e.g. the height of the robot and other types of behaviours made it the preferable testing platform for exploring the Reality Gap using the Transferability Approach.

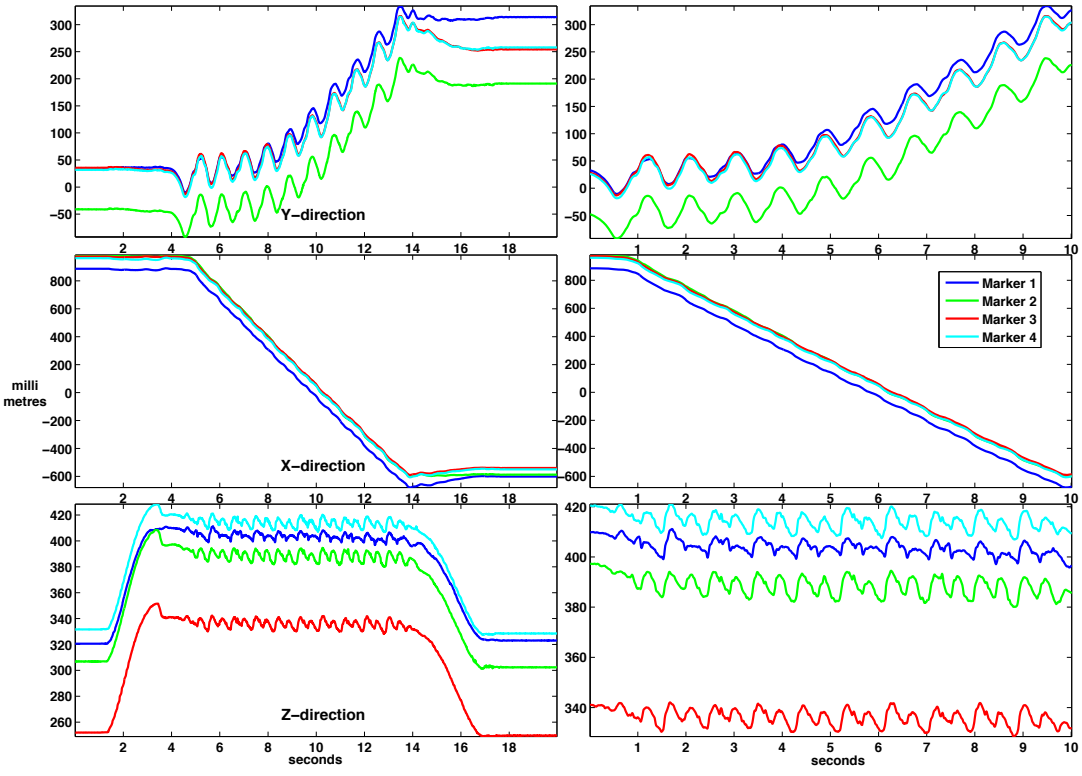


Figure 4.5: An example showing how to extract the evaluation period of a typical gait in the MoCap lab. First column: The 3-dimensional movement of the Nao robot using evolved gait parameters from a simple GA captured using four distinct passive markers. Second column: The extracted ten seconds long evaluation period.

4.4 Evolutionary optimisation experimental setups

For all the locomotion experiments, the top three parameters listed in Table 3.1 on page 54 have been kept the same, to make the results consistent and comparable to each other: The velocity along the x-axis has been set to 0.8 m/s, while the velocity in y-direction and around the z-axis have both been set to 0 m/s. This have made the focus to be solely on the gait altering parameters.

4.4.1 Preliminary investigation of walking gait optimisation using a simple GA

The preliminary experiment's setup using a simple Genetic Algorithm with an elitism rate of 0.10 to test out walking gait optimisation can be seen in Table 4.1. It is worth mentioning that the elitism in this experiment works by randomly selecting 10 percent of the current population for survival into the next generation.

Note that since this experiment does not utilise the NSGA-II, the mutation and crossover operators are different (see chapter 3.3.2 for explanations). The number of individuals in a population and the number of generations are also higher than for the succeeding experiments.

No. of individuals:	**200**	Elitism:	**Yes** (rate=0.10)
No. of generations:	**275**	Objective(s) (fitness):	**1. Distance walked**
No. of parameters:	**7**	Parameter initialisation:	**Random seed**
Crossover rate:	**1.00**	Crossover:	**Single-point (discrete)**
Mutation rate:	**0.25**	Mutation:	**Uniform**
Evaluation time:	**10 s**	Selection:	**Rank-based**

Table 4.1: The experimental setup for the preliminary investigation using a simple GA for gait optimisation.

4.4.2 Walking gait optimisation using NSGA-II

The experiment using a simple Genetic Algorithm with elitism (NSGA-II) with one objective for optimisation of a fast walking gait can be seen in Table 4.2 on the next page. The number of individuals is set to 40, and the evaluation has been set to run for 100 generations. These two numbers should be large enough to evolve good gaits, and is inspired by the numbers found in [44].

No. of individuals:	**40**	Elitism:	**Yes** (rate=1.00)
No. of generations:	**100**	Objective(s) (fitness):	**1. Distance walked**
No. of parameters:	**7**	Parameter initialisation:	**Random seed**
Crossover rate:	**1.00**	Crossover:	**Simulated binary (SBX)**
Mutation rate:	**0.30**	Mutation:	**Polynomial**
Evaluation time:	**10 s**	Selection:	**Binary tournament w/ crowded comparison operator**

Table 4.2: The experimental setup for the NSGA-II with one objective.

4.4.3 Walking gait optimisation using the Transferability Approach

The Transferability Approach's setup about finding fast walking gaits that transfer well from simulation to reality, i.e. copes with the reality gap can be found in Table 4.3. Notice the second objective, making this an Evolutionary Multi-objective Optimisation (EMO) problem.

No. of individuals:	**40**	Elitism:	**Yes** (rate=1.00)
No. of generations:	**100**	Objective(s) (fitness):	**1. Distance walked** **2. Transferability score**
No. of parameters:	**7**	Parameter initialisation:	**Random seed**
Crossover rate:	**1.00**	Crossover:	**Simulated binary (SBX)**
Mutation rate:	**0.30**	Mutation:	**Polynomial**
Evaluation time:	**10 s**	Selection:	**Binary tournament w/ crowded comparison operator**

Table 4.3: The experimental setup for the Transferability Approach using the NSGA-II with two objectives.

4.4.4 Walking gait optimisation on uneven terrain

The experiment using a simple Genetic Algorithm with elitism (NSGA-II) with one objective for optimising stable walking gaits on uneven terrain can be seen in Table 4.4. The lower number of individuals in each generation (32) was chosen to compensate for a longer evaluation period, as this more time-consuming evaluation could make the simulator unable to proceed to the next generation. It is unclear why this was a problem, as it only happened some times.

No. of individuals:	**32**	Elitism:	**Yes** (rate=1.00)
No. of generations:	**100**	Objective(s) (fitness):	**1. Distance walked**
No. of parameters:	**7**	Parameter initialisation:	**Random seed**
Crossover rate:	**1.00**	Crossover:	**Simulated binary (SBX)**
Mutation rate:	**0.30**	Mutation:	**Polynomial**
Evaluation time:	**15 s**	Selection:	**Binary tournament w/ crowded comparison operator**

Table 4.4: The experimental setup for the NSGA-II using one objective on uneven terrain.

Chapter 5

Results and analysis

A listing of all the conducted experiments and their approaches can be seen in Table 5.1 on the following page. The top one is a preliminary investigation into doing evolutionary optimisation of bipedal locomotion on the Nao platform, and the result itself is for that reason not too important. The next three experiments uses the NSGA-II algorithm with one objective, optimising fast walking gaits in the Webots simulator in the first one, before transferring and testing them out on the real robot on two different surfaces. Experiments 5-8 uses the Transferability Approach to deal with the Reality Gap problem, and the resulting gaits have been tested out on the real robot as well. The second to last experiment in the list is about optimising gaits on an uneven terrain in simulation to make for a more stable walk, and the last one shows the results when transferred onto the real robot on a carpet floor.

No.	Approach	Environment	Workload	Objectives
D1.	Preliminary investigation - Simple GA	In simulator	200 individuals/ 275 generations	1. Distance walked
D2.	Simple GA with elitism (NSGA-II)	In simulator	40 individuals/ 100 generations	1. Distance walked
D3.	Simple GA with elitism (NSGA-II)	On laminated floor	5 runs / solution	Transfer and test 3 best and 3 worst solutions in the last gen (from 2.)
D4.	Simple GA with elitism (NSGA-II)	In MoCap lab (carpet floor)	5 runs / solution	Transfer and test 3 best and 3 worst solutions in the last gen (from 2.)
D5.	Make transferability function	In simulator and on laminated floor	5 runs / solution	Using the results from 2. and 3.
D6.	Transferability approach (NSGA-II)	In simulator	40 individuals/ 100 generations	1. Distance walked 2. Transferability function
N7.	Transferability approach (NSGA-II)	On laminated floor	5 runs / solution	Transfer and test 6 different solutions from the Pareto front (from 6.)
D8.	Transferability approach (NSGA-II)	In MoCap lab (carpet floor)	5 runs / solution	Transfer and test 6 different solutions from the Pareto front (from 6.)
N9.	Simple GA with elitism (NSGA-II)	In simulator on uneven terrain	32 individuals/ 100 generations	1. Distance walked
N10.	Simple GA with elitism (NSGA-II)	In MoCap lab (uneven terrain)	5 runs / solution	Transfer and test 3 best and 3 worst solutions in the last gen (from 9.)

Table 5.1: Overview of all experiments conducted in this project.

5.1 Preliminary investigation - Using a simple GA with elitism in simulator

This experiment has utilised the approach described in Chapter 4.2.1 on page 56 using the experimental setup described in Table 4.1 on page 66. As one can see in Figure 5.1 on the next page, in the start of the optimisation, the best individual in each generation sees a rapid increase in fitness. After around 65 generations, the fitness value has almost been doubled. However, it seems like the good fitness values in solutions obtained in one generation have a tendency to not be brought over to the proceeding generations. This is effectively making one not having any guarantee of avoiding a decrease in solution quality over time, making the performance of the gait optimisation in this experiment very poor. Maybe the most dramatic effect for this specific evolutionary optimisation can be seen after

188 generations, where the best solution of the whole process is found: Even though one achieves a fitness value of over 1.13 metres after this number of generations, the best individuals obtained in the remaining populations do not even get close to such fitness values, ending up with a fitness value of just over 0.7 metres in the last generation, showing a decrease of almost two thirds compared to the best obtained solution. This value is also only about 0.2 metres higher than the fitness obtained during the first generation, resulting in almost no increase at all in fitness value during the whole run. This poor result can be an indication of something wrong with e.g. the implementation or the setup of the Genetic Algorithm.

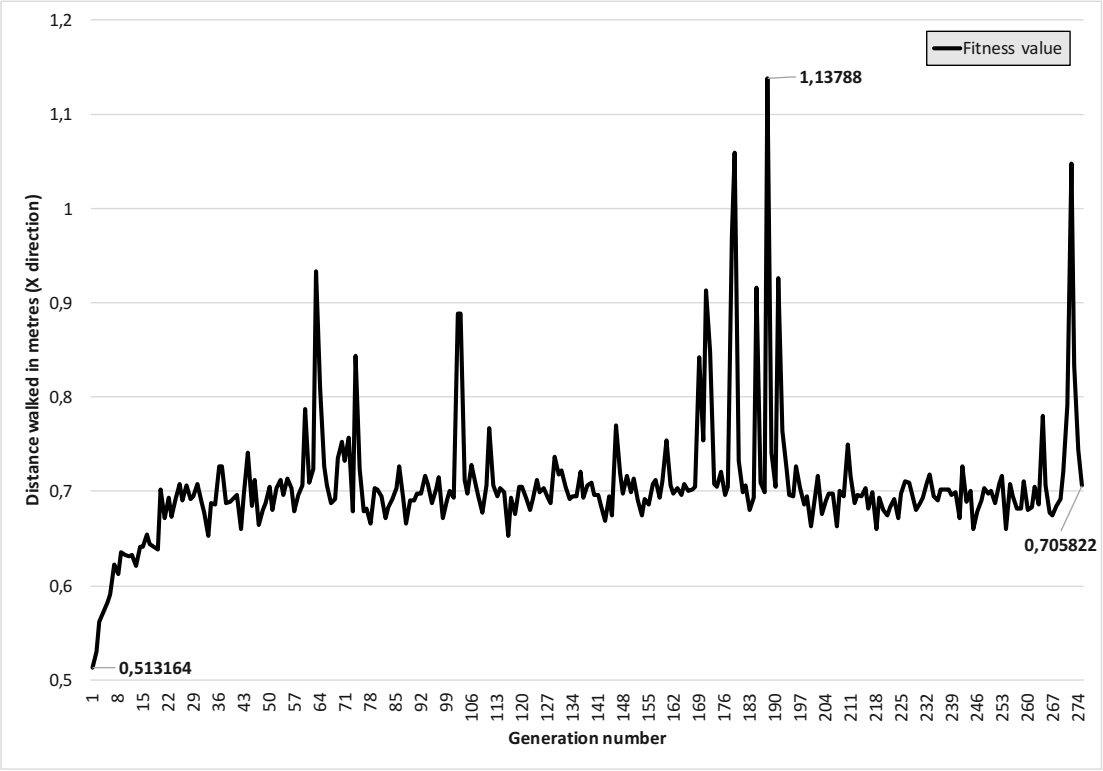


Figure 5.1: The evolution of best fitness values obtained using a simple Genetic Algorithm in simulator. Included is the best fitness value found in the first, best and last generations.

5.1.1 Analysis

The reasons for this can be many, and it was at a later stage in the process discovered that the time in the simulator was not properly synced, making this a potential source of error. The simulator has a feature allowing the simulation time to be synced with the time accessible by the robot controller, ensuring e.g. the evaluation time to be precisely the same for all trials. Enabling this feature was done after this experiment was conducted, and can possibly explain parts of this non-deterministic

behaviour. Another discovery was that the robot was not properly reset after every individual, making the robot perform worse in evaluations being done after a fall. The reason for this was that if the robot fell on the side, the corresponding arm was pushed closer to the body, making them too close after some falls on each side, effectively using them to push the legs back and forth from behind while walking. This could in turn make the robot fall over more often due to the robot's feet starting to swing, making the problem of falling even worse. This can also have impacted the results of the optimisation, as later experiments have been able to obtain better fitness values in a smaller number of evaluations.

To fix the problem of properly resetting the robot, it was decided to use the NAOqi Framework's own method for resetting the posture of the robot. By doing it this way, the robot's joints, such as in the arms, legs and hip would be reset to the same start positions before every evaluation, increasing the robustness of the trials. This was found to be especially useful when conducting trials on the real robot, as lifting and placing the robot back to its start position could alter the positions of some of these joints. It was also decided to use the Webots simulator's built-in function to reset the whole simulator between every generation, removing any possible sources of error due to long simulations and making the start of every generation as similar as possible. A function in the simulator to place the robot in the same start position before every trial was also utilised, even though the terrain was supposed to be the same no matter the start point. (It is worth noting that these fixes were implemented after this experiment was conducted, and for that reason has not had any impact on the following experiments.

The mentioned time syncing problem was present in the experiment as well, but was corrected before the other experiments, and has therefore not had any impact on their results. It is worth mentioning that although this result can seem non-deterministic only due to the time being not synced correctly, as well as lacking a proper reset before each trial, it is reasonable to assume that the previously mentioned elitism rate of 0.1 can also have impacted the result, as it randomly selects the decided rate of the current population for survival into the next generation. Due to this strange implementation, it was decided to use the NSGA-II algorithm for the rest of the experiments, even when optimising only one objective.

The reason for discarding this randomly selecting elitism operator is that the syncing error should not only make the fitness of the solutions worse than what they really are due to a shorter evaluation period, but also the opposite, i.e. making some have a higher fitness than what is deserved. If the GA utilised a higher elitism rate, e.g. full elitism with a rate of 1.0, not only solutions that perform well would have survived into further generations, but also the cheating ones, not being evaluated again, but kept due to this one evaluation only. If done this way, one would expect to see some unlikely high fitness values due to the good solutions that have been evaluated for a too long time. Instead, when one has this kind of

elitism, the solutions are therefore not improved that much, due to the high survival rate of the currently evaluated solutions, and the overall curve of the evolution of fitness values is therefore flatter than one would normally expect from a GA.

5.2 NSGA-II results in simulator

This experiment has utilised the approach described in Chapter 4.2.1 on page 56 using the experimental setup described in Table 4.2 on page 67. A plot with the result of this evolutionary optimisation can be found in Figure 5.2 on the following page.

5.2.1 Analysis

Since the NSGA-II incorporates elitism even if one optimises only one objective, one is guaranteed the survival of good solutions into later generations. Because of this, the presented result is more or less expected. The slope of the curve can maybe be said to be the result of a classic GA optimisation problem, where one sees a rapid improvement in the beginning after a random initialisation using the whole search space. As the fitness is improving, the search space is moved towards either a local or a global minimum, i.e. places with high fitness, and the improvements in fitness occur less often than before. Since the default parameters are solely concentrated with stability in mind, the results obtained already during the first generation performs better than the default ones. The solution with the best fitness value is found after around 40 iterations, even though the rest of the population continues to improve. The reason for running the evaluation even after the improvement of the best individual had stopped was in fact due to one wanting the whole population to improve a lot, and not only the best individual. The reason for this is found in section 5.5 on page 78. When it comes to the performance of the best individual, the fitness value it obtained during the evaluation had a score of 1.38 metres. The parameters for the genotype representation for this specific solution are listed in Table 5.2, together with the default parameters.

MaxStepX	MaxStepY	MaxStepTheta	MaxStepFreq	StepHeight	TorsoWX	TorsoWy
0.068	**0.112**	**0.217**	**0.750**	**0.017**	**0.036**	**0.054**
0.040	0.140	0.349	1.000	0.020	0.000	0.000

Table 5.2: The best solution obtained using the NSGA-II with one objective in simulator in the top row, together with the default parameters.

As one can see in the table, maybe the most important parameter is MaxStepX, which decides the maximum allowed forwards movement in

metres. As mentioned earlier, this parameter can vary between 0.001 and 0.080, although the developers recommends it not to be larger than 0.060 metres, due to instability on soft surfaces. Since the optimised solution is larger than this, it will be interesting to test this solution out on surfaces such as laminated and carpet floors. Experiments on this are described in the two following sections, along with comparisons between different solutions in simulation and on the real robot. It is also worth noticing that the maximum allowed distance in y-direction and the rotation along the z-axis are smaller than for the default parameters, most likely due to the fitness function subtracting any sideways movement. The MaxStepFrequency parameter is also a bit smaller, probably due to the longer steps being taken requiring a less rapid frequency of steps to make the walk stable. The StepHeight parameter, i.e. the maximum allowed upwards elevation for each step is also probably smaller to make for a faster and more stable gait. It will be interesting to see if the value of this parameter will be a lot different when optimising a walk on a more uneven terrain. This experiment can be found in section 5.9 and 5.10.

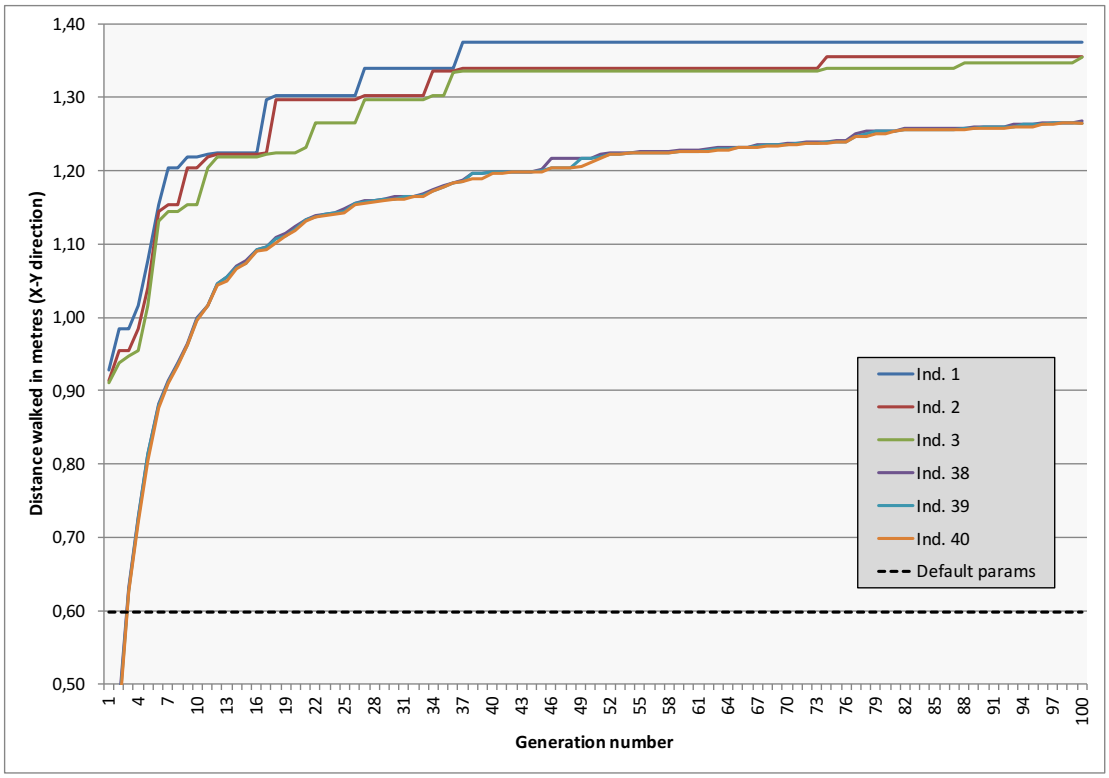


Figure 5.2: Evolution of fitness values obtained using NSGA-II with one objective in simulator.

5.3 Gaits using evolved parameters from NSGA-II GA on laminated floor

To test out how the gait solutions found in the previous section actually performs on the physical Nao, an experiment making the physical Nao walk using a number of the evolved solutions from the final population was conducted. A chart showing a comparison between the performance of these gait parameters in the simulator and reality can be seen in Figure 5.4 on page 79. The figure also contains the fitness values obtained in an experiment using a carpet floor, which is more described in the next section. The gait parameters have each been run 5 times, both in the simulator as well as on the laminated floor, to make the results more robust.

5.3.1 Analysis

As seen in the figure, the performance of the transferred gaits are seemingly good, and more similar than when transferring the default ones. The best parameters however, depends on what the main goal is: When it comes to the gait that seems the most transferable, individual number 15 is clearly the best, with very little difference in the walked distance in simulator and on the laminated floor. Individual number 25 makes on the other hand the robot walk the longest during the evaluation period, with almost 1.3 metres in 10 seconds. This results in a forwards speed of around 0.13 metres/second, something which compared to the default speed of 0.11 metres/second using the same approach actually is a bit faster. The recently published study[64] that allows for a variable robot height however has made the robot walk in a speed of around 0.34 metres/second. The similar DARwIn-OP's has a faster top speed of 0.24 metres/second, but this robot is a bit shorter, but also quite lighter.

An overall impression one gets by analysing the results, is that the robot moves sideways when using the real Nao. By observing the different walks, it seems that the robot is walking more sideways in the end of the run, slowly turning to the left. If one takes a closer look at the different walks, the average sideways movement is much higher when using the physical robot, compared to in the simulator. The average movements in x- and y-direction over 5 runs for some selected gaits can be seen in Table 5.3 on the following page. It is unclear whether the laminated floor impacts the walk by making the robot slip or move sideways, or if there are other reasons for this unwanted effect.

Gait parameters	Webots(X)	Webots(Y)	Real(X)	Real(Y)
Default	0.609	0.021	0.876	0.146
Ind. 3	1.249	0.037	1.552	0.292
Ind. 15	1.242	0.071	1.540	0.372
Ind. 25	1.237	0.080	1.610	0.328

Table 5.3: The average movements over 5 runs for some selected gaits evolved using the NSGA-II with one objective in simulation, compared with the default one, on laminated floor.

The table shows the walked omnidirectional distances averaged over 5 runs for the seemingly most similar individual (number 15), together with the best performing individual (number 25), as well as the parameters that had the smallest sideways movement (individual number 3). This table clearly show a reality gap between the sideways movement in simulator and reality. Some of it can come from measurement errors, as there is a difference between forwards movement as well, although not that large. Even for individual 3 with the lowest sideways walk, the movement in y-direction is nearly 8 times longer when using the real robot. Another source of error can be the surface, as a laminated floor can be quite slippery. The same gaits have therefore also been tested on the robot using the carpet floor found in the MoCap lab. This experiment is more described in the next section, comparing the robot's performance on the different surfaces.

Another interesting thing that in fact shows more clearly the problem with having a Reality Gap, is pictured in Figure 5.3 on the next page. The figure contains a plot with differences in fitness values between simulator and reality as the fitness increases. The values are gathered from the previously mentioned runs done in simulator and on the laminated floor, together with a line showing the ideal relationship, having no difference at all in performance, i.e. no Reality Gap. As the graph points out, it seems like a higher fitness value can lead to a larger Reality Gap. This result is rather expected, as the parameters giving the highest fitness values often exploits features not modelled well in the simulator, and is most likely the case also in this experiment. This is as explained in chapter 2.2.1 on page 11, and has given strength to the plan of doing experiments that deals with this problem. A variant utilising the Transferability Approach for this problem can therefore be found in the chapters beginning with chapter 5.5.

5.4 Gaits using evolved parameters from NSGA-II on carpet floor

This experiment has been done using the physical Nao robot on the carpet floor found in the MoCap lab. A plot depicting the differences between the walked distances using the parameters optimised in section 5.2 is found in Figure 5.4 on page 79. It shows the average performance of the gaits after

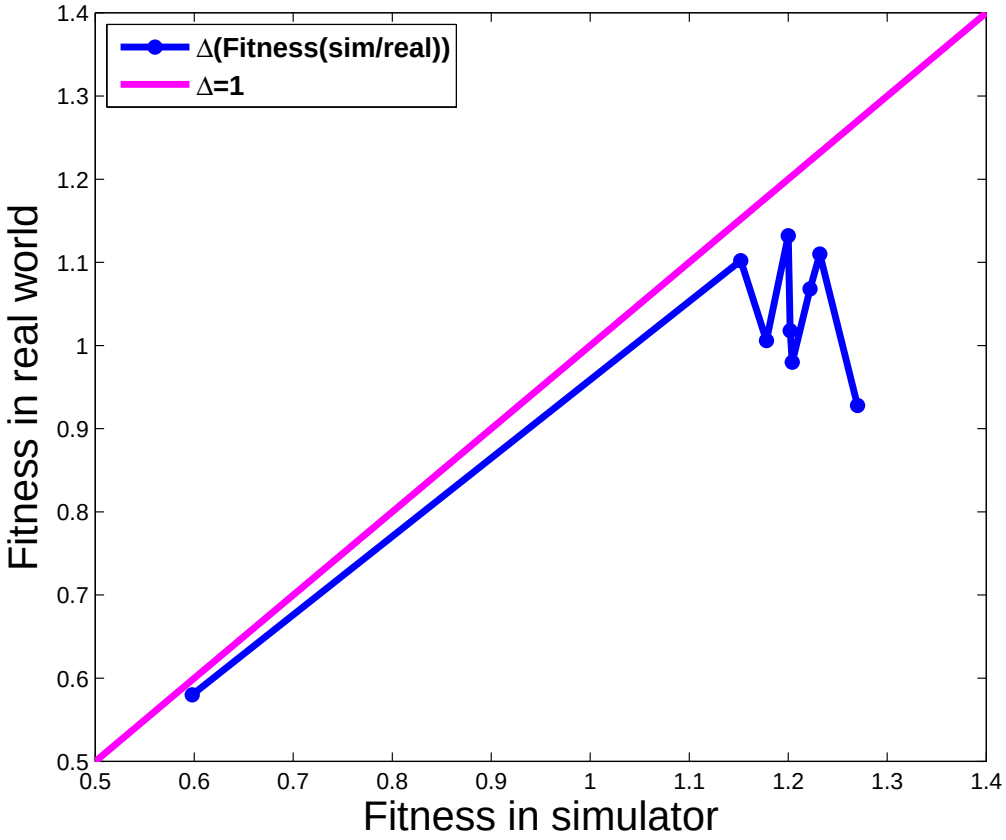


Figure 5.3: The difference quotient between fitness values in simulation and reality in the experiment described in section 5.3, as the fitness increases. Ideally, it will follow the blue line (Δ=1) for all fitness values, indicating no reality gap. As this is not the case here, the solutions with a higher fitness is likely to exploit features not modelled well in the simulator.

being run 5 times, together with how they performed in simulation and on laminated floor (sections 5.2 and 5.3). The performance measure, i.e. fitness function in this experiment is as it has been in the previous sections. The movements were captured by taking the average of the movement done by each of the four markers, making the measurements as accurate as possible. The presented result is not only important compared to the behaviour in simulator, but also compared to on laminated floor, to see if the previously mentioned slipping and walking out to the left comes from the surface, or if it is a Reality Gap problem.

5.4.1 Analysis

As an overall impression, it seems like some gaits performs better on the carpet floor, while some others are worse. The reason for this can easily be

investigated by looking at the x- and y-movements separately, as has been done in Table 5.4. The individuals chosen to present are the ones that have the closest fitness values (numbers 5 and 15), together with the one with the best fitness as well as the default ones.

Gait parameters	Webots(X)	Webots(Y)	Real(X)	Real(Y)
Default	0.609	0.021	0.845	0.046
Ind. 5	1.257	0.060	1.540	0.339
Ind. 15	1.242	0.071	1.492	0.352
Ind. 25	1.237	0.080	1.638	0.286

Table 5.4: The average movements over 5 runs for some selected gaits evolved using the NSGA-II with one objective, compared with the default one, on carpet floor.

By reading the table, the claim that there really is no significant difference in performance between the laminated and carpet floor is confirmed. The little difference found can probably be explained by measurement errors. This result together with the one in the previous section strengthens the argument to explore techniques that deal with the Reality Gap to an even greater extent, and is for that reason conducted and described in the succeeding sections.

5.5 Making a transferability function

As described in the experiments chapter, the most reasonable function to make for use in the Transferability Approach is to use an observed behaviour in simulation as input, and the difference in fitness values as output. The reason for doing it this way is because the mapping from parameters to a transferability would require seven input variables, making it a highly non-linear function. Since one only has data from transferring a limited number of gait parameters, the transferability function has to be made from these points using some kind of interpolation technique. Like in [52], it was decided to use the Inverse Distance Weighting (IDW) [65] for interpolating the points and to make the function. It works by assigning values to unknown points by taking a weighted average between the values found in the already known surrounding points. The resulting three- dimensional function is shown in Figure 5.5 on page 80. The points used to make the function were taken from the experiments described in chapter 5.2 and 5.3, as the difference in performance between laminated and carpet floor was small. The behaviours in the simulator were used as inputs to the function, i.e. the travelled forwards and sideways distances. The output was decided to be the differences in fitness values, using it as a transferability score. This is the same way of mapping the function as shown in Figure 2.4 on page 16. By looking at the function, it is easy to see that high fitness values makes for a poor transferability score, i.e. non-transferable gaits. However, it

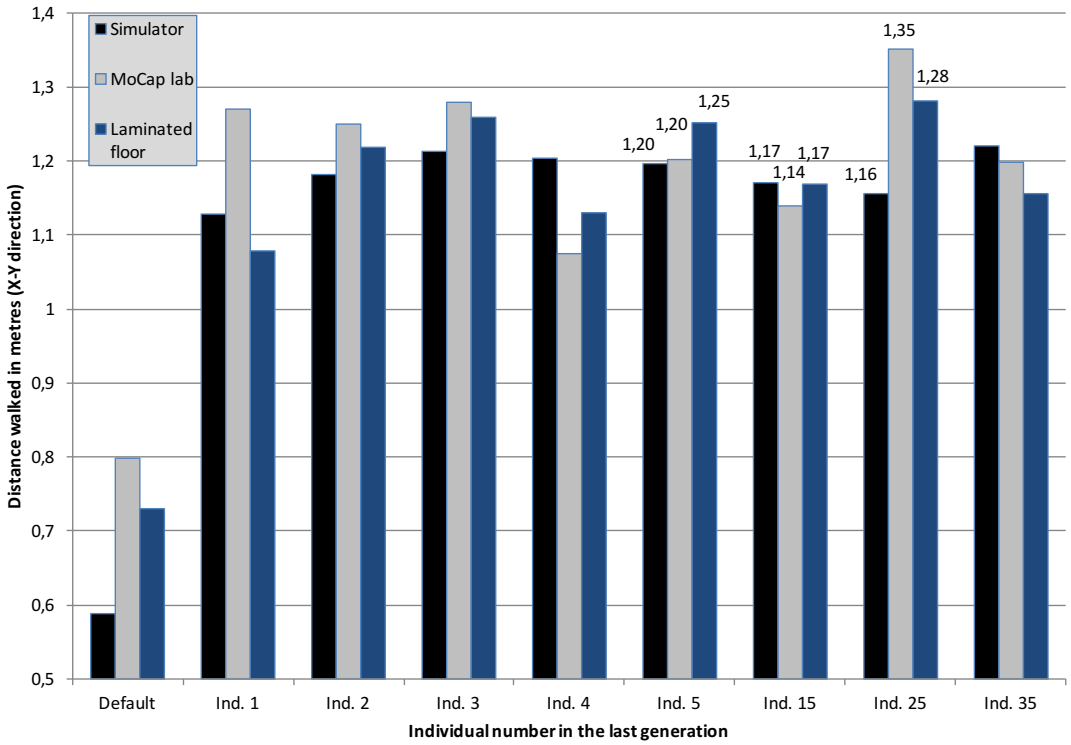


Figure 5.4: A combined plot showing performance differences between distances walked using evolved gait parameters from NSGA-II with one objective for some selected solution parameters between the simulator, laminated floor and carpet floor.

will be interesting to use it as a second objective together with the original objective for optimising walking gait patterns, to see if one is able to reduce the Reality Gap. This is done in the following sections.

5.6 Transferability Approach results in simulator

This experiment has utilised the approach described in chapter 4.2.2 on page 57 using the experimental setup described in Table 4.3 on page 67. The result of this Evolutionary Multi-objective Optimisation (EMO) of walking gaits using the NSGA-II with two objectives can be found in Figure 5.6.

5.6.1 Analysis

When optimising using two (or more) objectives, an easy way to visualise the final result is by plotting the Pareto optimal front, as explained in the EMO part of chapter 2.1.3. Here, all the blue points represent the so-called optimal solutions, due to the trade-off between their walked distances and

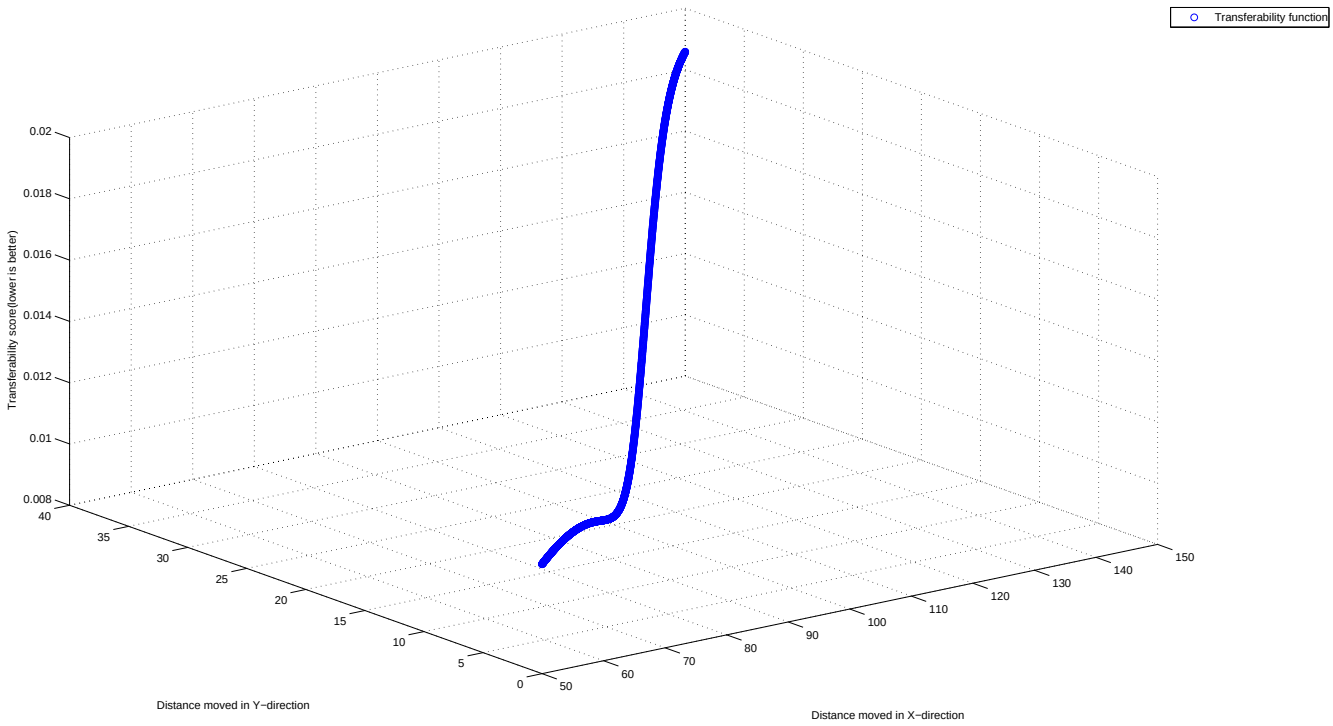


Figure 5.5: Interpolated transferability function using IDW Interpolation.

the transferability of them. These solutions are therefore placed in the first front, called the Pareto optimal front. The red points on the other hand are the solutions found in front number two; the so-called dominated front. These are dominated by the Pareto optimal solutions, meaning they perform worse in at least one of the objectives, although the other objective actually can be better. Solutions found in higher fronts are dominated by this second front, and have been chosen to not be shown in this figure. The ideal region for the solutions is placed in the bottom left corner, giving a result that consists of a long walked distance as well as having an optimal transferability from simulation to reality. This means that the evolution is able to find gaits that transfer well, even if there exists a Reality Gap problem. This is clearly not the case here, due to the previously observed Reality Gap between the Nao robot's behaviours in simulation and reality. As a consequence, one can expect the evolutionary optimisation process to not find gaits placed in this optimal point region, as this would only happen if the Reality Gap was not an issue. Instead, the EA needs to find gaits that perform well, i.e. walk fast, in addition to being transferable to reality at the same time. An example can be to set a maximal accepted transferability score, discarding all solutions over this limit.

When it comes to the actual performance of the gaits, compared to when only optimising using the first objective (chapter 5.2), the evolutionary

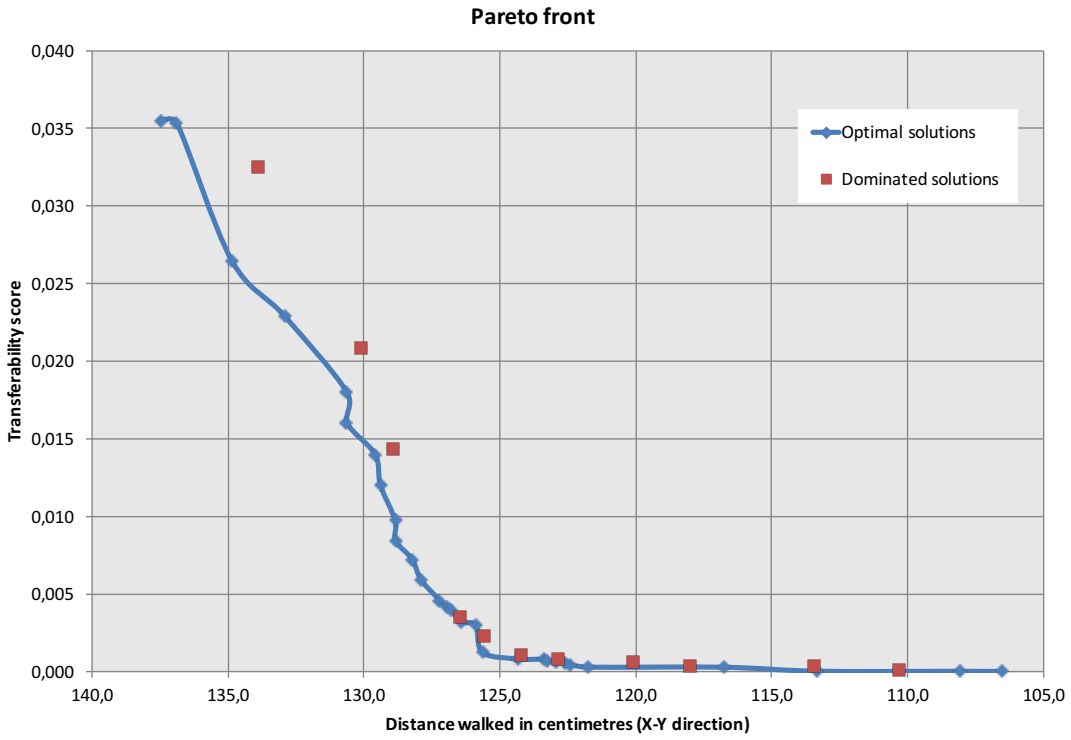


Figure 5.6: Results obtained with the Transferability Approach using the NSGA-II in the Webots simulator. The first axis (bottom) shows the walked distance in cm over the evaluation period (10 seconds) in X-Y direction, while the second axis shows the transferability score, where a lower value means a better transferability from simulator to reality.

process also this time does achieve similarly high fitness values during the same number of generations. These however have a quite high transferability score, seemingly making them less transferable than others with shorter walked distances. This is more or less expected, as the good solutions from the first approach had a Reality Gap problem when being transferred, resulting in the transferability function giving them a poor performance in the second objective, i.e. having high transferability scores. Since all of the gaits in the Pareto optimal front have a similar trade-off between the two objectives, it is not easy to tell which of them to transfer to reality, nor to decide a winner among them. It all boils down to what one wants to achieve: If a fast walking gait is the most important, the solutions in the left part of the figure are desirable. If the minimisation of the Reality Gap is the most important factor, the further down in the figure one goes, the less Reality Gap one should expect. These are probably the desirable solutions, and a physical experiment involving transferring solutions from both parts of the front on to the real Nao robot is described in the section 5.8.

A thing worth mentioning was that the robot had a tendency to fall over more often during the evaluations in the simulator. An explanation of this can be that since the evolutionary process now optimises the gaits to not only be fast and stable, but also transferable. This can make the search move towards seemingly very transferable solutions that can turn out to be unstable, causing the robot fall over, making them end up being sub-optimal.

5.7 Transferability Approach gaits on laminated floor

Although the measurements used to make the transferability function were done using the Nao robot on laminated floor, conducting this experiment was found to be unnecessary. The reason for this decision was due to the similar behaviours obtained when walking on both laminated and carpet floors, as described in chapter 5.4. The advantages of using the MoCap equipment such as more accurate results in addition to being less time-consuming were used as arguments for making this decision. The experiment testing out the gaits on the carpet floor is described in the next section.

5.8 Transferability Approach gaits on carpet floor

The previous experiments already have resulted in fast and stable walking gaits, although they have a tendency to walk further and further out to the left. Therefore, the most interesting part of this experiment is rather to consider if the Transferability Approach with its function has improved this part of the gaits in particular, as well as other parts of them. The individuals chosen to be transferred onto the real robot were the two with the lowest best fitness values for each of the objectives, in addition to six others randomly selected from the ones having a transferability score under 0.010.

A comparison of the robot's walked omnidirectional distances, i.e. the forwards movement subtracted by any sideways walking between in simulation and reality, can be found in Figure 5.8 for different gait parameters represented by individuals found in the last generation of the optimisation. When comparing this graph with the one that used the NSGA-II with one objective to optimise gaits (Figure 5.4), it looks at a first glimpse like the Transferability Approach has made the Reality Gap rather larger than smaller, contrary to the intention. Most of the solutions look quite off, with individuals like number 4, 15, and 35 having a better performance in reality, while numbers 2, 3, 5 and 25 perform worse. The best gait parameters, represented by individual number one however seems similar, and should have the best trade-off between fitness and transferability, but by looking at Figure 5.8, this is actually not the case.

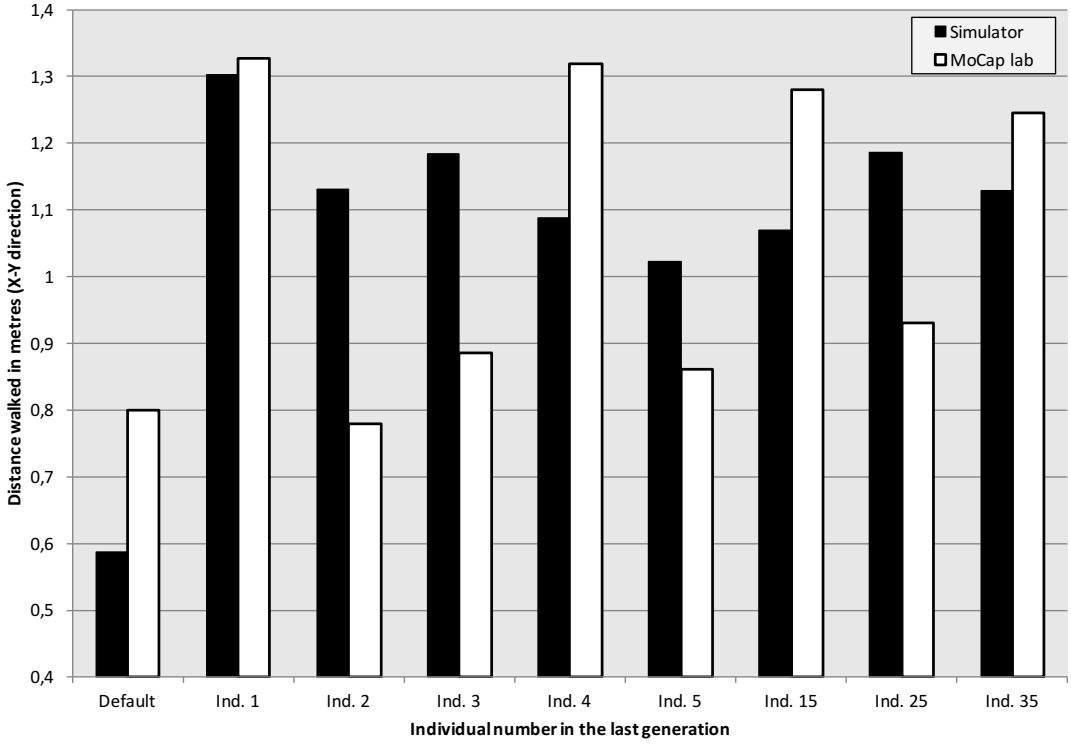


Figure 5.7: Difference in gait performance between walked distances using evolved gait parameters from Transferability approach in MoCap lab and in simulation.

5.8.1 Analysis

Figure 5.8 shows a comparison of some selected gait parameters evolved using the NSGA-II with one objective described in section 5.2 and found in Figure 5.4, together with some solutions evolved using the Transferability Approach, found in Figure 5.8. It has unlike these previous charts, distinct columns for the movements in x- and y-directions, measured five times for each parameter set both in simulator and on the carpet floor in the MoCap lab. By looking at the top chart using only the one objective, one can see that the difference in forwards movement seems fairly large for all the different individuals, i.e. between the green and purple columns. The sideways movement is also much longer in reality than in simulation, with values ranging from 0.3 to 0.4 metres during the whole evaluation period, clearly a lot more than when tested in the simulator. This is represented by the blue and orange columns.

The bottom plot shows the same number of individuals, but now representing gait parameters optimised using the Transferability Approach. Some of these solutions achieve similar performances in x-movement compared to the ones in the top plot, although the overall y-movements seem lower. The

previously mentioned individual one that looked similar in Figure 5.8, has when looking at the omnidirectional distances separately, actually quite a large Reality Gap. Despite that, there are also parameters that have almost identical performances in the covered forwards distances, such as individual number 2, 3, 5 and 25. They have a much smaller Reality Gap when considering the walked x-distances than when not using the Transferability Approach. Unfortunately, the problem with the robot walking more and more to the left during the runs is still apparent for these solutions, although the differences seems to be smaller.

A more detailed plot containing only these selected solutions can be found in Figure 5.9. As one can see, the differences in movement in the x-direction are small, although there still exists a significant difference in y-movement. It therefore seems like the Transferability Approach has a tendency to be able to reduce the existing Reality Gap in forwards movement by a large amount, as well as reducing the problem with sideways movement, although still apparent.

It is worth mentioning that this Reality Gap problem is still apparent when using the default parameters. As shown in Figure 5.8, the simulation of this gait shows no significant sideways movement. This might suggest that the problem is connected to the specific Nao robot, as there has not been found any documentation indicating this behaviour to be a general problem. Another interesting thing in this figure is that individuals 25 and 35 seems to have a small, but negative sideways movement in the simulator, although this is not the case on the real robot. A negative y-value means that the robot walks to the right (See Figure 5.11 for the axis definitions). The problem with a large sideways movement is therefore still there when using these kind of individuals.

5.9 Gaits evolved on uneven terrain in simulator

This experiment has utilised the approach described in chapter 4.2.3 on page 59 using the experimental setup described in Table 4.4 on page 68. A plot showing the evolution in fitness values in simulation for all generations using the NSGA-II with one objective on uneven terrain can be found in Figure 5.10.

5.9.1 Analysis

When comparing this result with the ones evolved on a flat surface found in chapters 5.2 and 5.6, the best fitness values obtained using this approach have values that are much smaller, even though the evaluation period is 5 seconds longer. A reason for this can probably be that the speed when walking up and down needs to be smaller to not make the robot fall over. Another reason are the fact that the actual distance covered are longer than

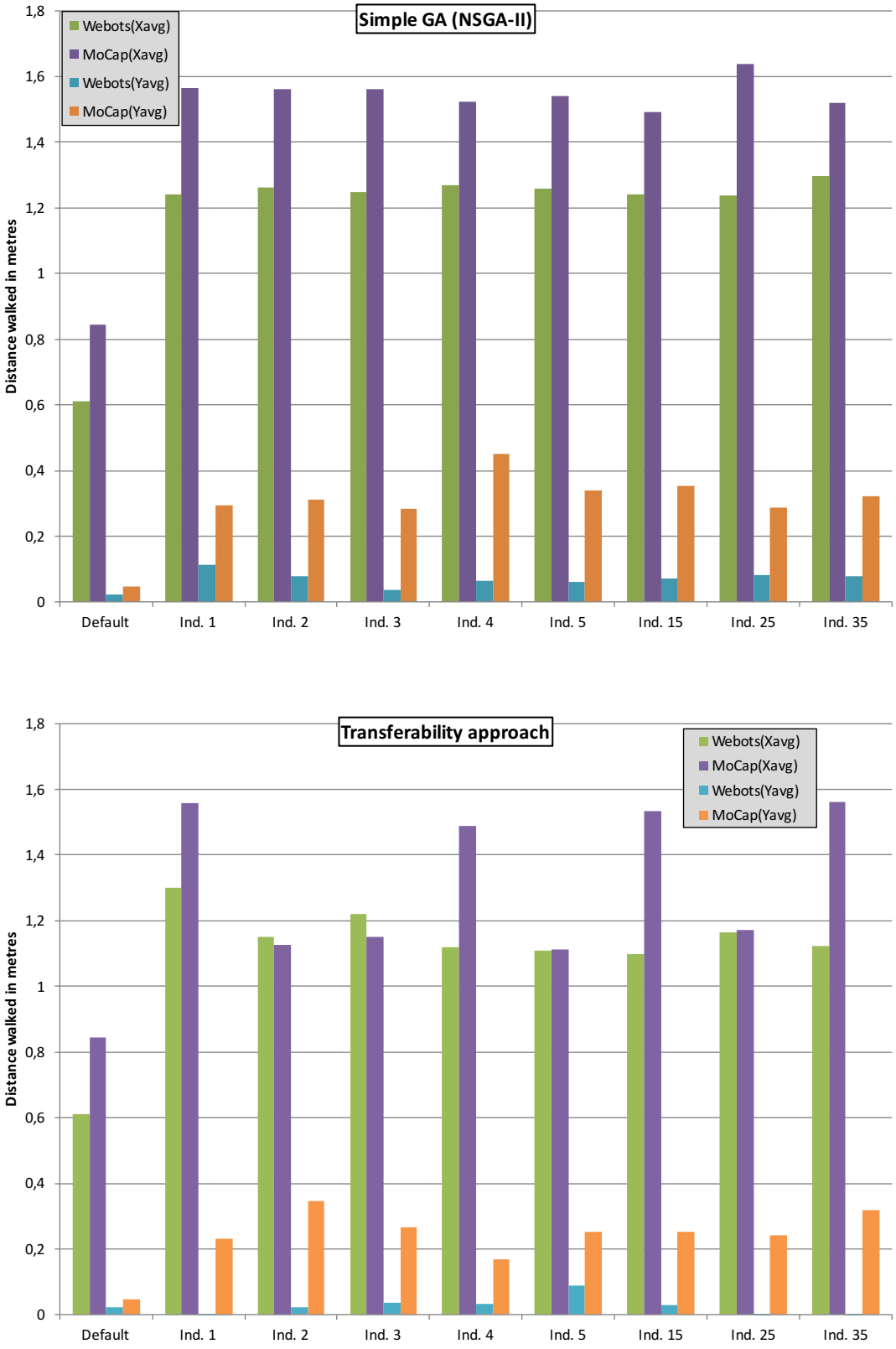


Figure 5.8: Comparison of walking gait distances in x- and y-direction evolved using two different approaches. The horizontal axes shows the solution numbers in the last generation.

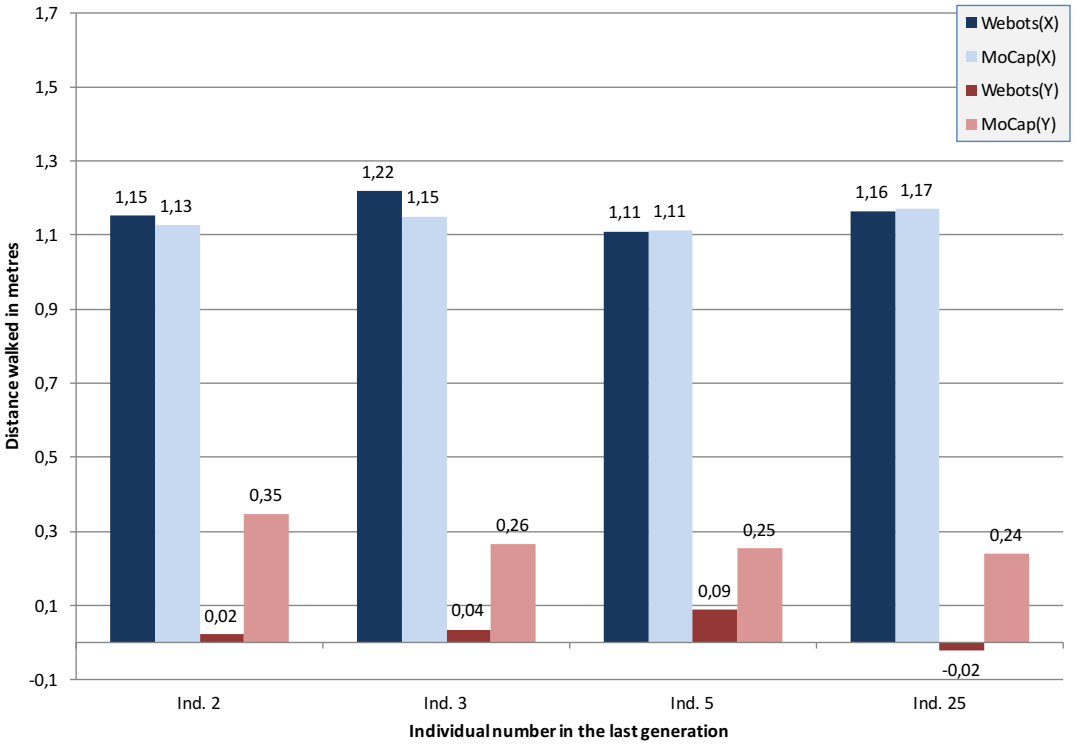


Figure 5.9: The best solutions found using the Transferability Approach.

when measured purely in x- and y-direction. This is because the robot also moves along the Z-axis, i.e. upwards and downwards, which is not the case when walking on a flat floor.

The evolution of fitness values in this case does not have as rapid increase as was the case for the previous approaches. This is probably due to the harder optimisation problem. A side-effect of this is that the best individual in each generation keeps improving for many more generations as well, although the fastest increase also here happens during the first 50 generations. These are the two reasons for running the evaluation for a larger number of generations, compared to the previous experiments. When it comes to the resulting best fitness values, the three best individuals achieved similar values of about 0.75 metres during the 15 seconds evaluation period. The last individual on the other hand came close with a value of little under 0.7 metres. This is one of the disadvantages of having such as small population number: The parameters and fitness values tends to get close to each other, decreasing the possibilities of finding radical new solutions that perform better. All of the individuals however performs better than the default ones, as the robot falls over when it starts to climb the first ramp.

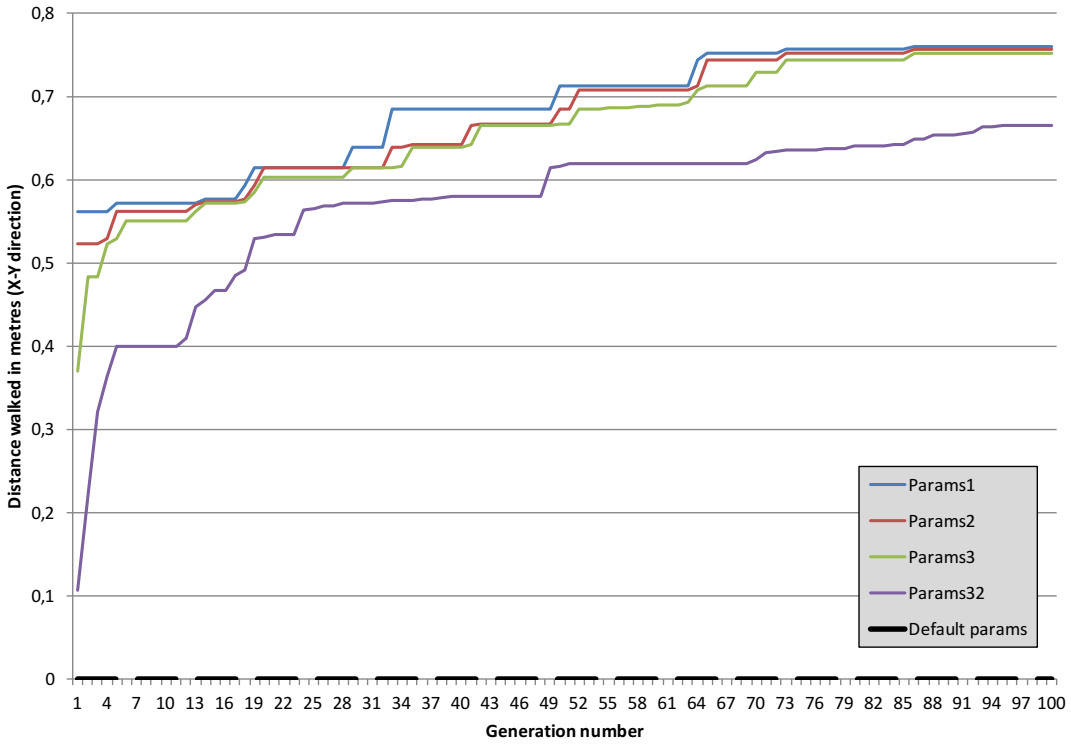


Figure 5.10: Evolution of fitness values obtained using NSGA-II with one objective on uneven terrain in simulator.

MaxStepX	MaxStepY	MaxStepTheta	MaxStepFreq	StepHeight	TorsoWX	TorsoWy
0.068	0.112	0.217	0.750	0.017	0.036	0.054
0.038	**0.139**	**0.075**	**0.946**	**0.016**	**0.005**	**0.122**
0.040	0.140	0.349	1.000	0.020	0.000	0.000

Table 5.5: The best solutions obtained using the NSGA-II on even and uneven terrain in simulator, shown in the second and third rows, respectively. The bottom row shows the default parameters.

A comparison of the best individuals' gait parameters on both uneven and even terrain can be found in Table 5.5, along with the default parameters that turned out to not work on this uneven surface. As mentioned in the previous section, the uneven terrain-optimised gait walks slower than it's flat counterpart. A third explanation can be found in the MaxStepX parameter, which is much smaller and in fact close to the default value. This value represents the maximum allowed forwards step length, and can be a sign of necessity of taking small steps when walking e.g. uphill. This is as expected in chapter 4.2.3. These short steps should also come more rapidly, as the maximum step frequency value is around a quarter larger, also near the default value. The MaxStepY parameter is a little larger than

for the even terrain optimisation, allowing the feet to take wider steps, probably to make it more stable. The fast gait optimised on even terrain on the other hand most likely has found this value to be smaller because walking on a flat surface should require less stability, and at the same time taking more narrow steps should increase the speed, as more power is used to make the robot move forwards. The little MaxStepTheta value compared to the default and even terrain optimised walks, allows for only a small rotation around the z-axis for the robot. The reason for this is likely because a large rotation can make the robot end up falling over the side edges of the ramp, something that would not be the case during the more simple optimisation environment. The StepHeight parameter, i.e. the maximum allowed elevation along the z-axis is a little smaller than the default value, almost identical to the value evolved on the flat surface, and much lower than the maximum allowed number. The most logical explanation of this is that taking high steps will probably slow down the gait and therefore not get a high fitness value when optimising. It is nevertheless interesting that this is the case, as one would perhaps expect a high step height value when walking uphill.

The most interesting parameters however, are the two concerning the maximum allowed rotations of the torso around the x- and y-axes. The default parameters do not allow for any torso rotations, while the values obtained when doing evaluation on the even terrain allows for rotations of about 2 and 3 degrees around the x- and y-axes, respectively. The resulting parameters in this approach however, allows for almost no rotations around the x-axis, with a value of under only 0.3 degrees. This means that the torso, i.e. the upper part of the robot body should not be able to move or lean much from side to side nor up and down, possibly minimising the chances for any oscillating behaviour, which in turn can lead to the robot becoming unstable, making the robot fall over. As for the maximum allowed torso rotation around the y-axis, the optimised gait has in fact the highest allowed value of almost 7 degrees. This means that the torso while walking can be rotated to lean forwards with a quite large angle. The reason for this is probably to keep the robot in balance while walking on a surface with a gradient, compensating the instability than can occur when e.g. entering the first ramp pictured in Figure 4.1 on page 60. A picture of the Nao robot with the different axes and angles can be found in Figure 5.11 on the next page. As one can see, having a slightly positive rotation around the red X-axis, i.e. having a small positive TorsoWx parameter value means that the robot's torso movement is able to just marginally move from side to side or up and down. The maximum value of the TorsoWy parameter on the other hand gives the robot's torso the abilities to rotate around the green Y-axis, e.g. to make it lean forwards to a much greater extent.

Because this experiment not only was intended to make the robot able to walk on a more uneven terrain, but also to make a more stable walk, testing out the gaits on a physical Nao robot on both similar ramps as well as on a flat surface should be interesting. An experiment using the evolved gaits on a flat carpet floor is described in the next section.

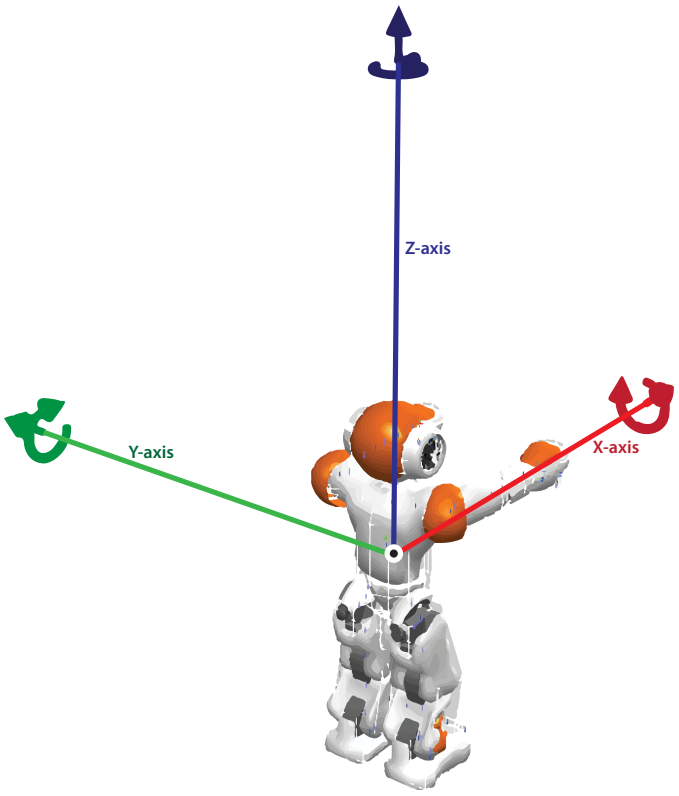


Figure 5.11: The Nao humanoid robot with its positive axes (the straight lines and arrows) as well as positive rotations (the circling arrows).

5.10 Gaits evolved on uneven terrain in MoCap lab

The gait parameters evolved on an uneven terrain in simulation was tested on the flat carpet floor in the MoCap lab, to see how well they performed, especially bearing in mind the gait stability. The result of this can be found in Figure 5.12. It was decided to transfer and test out the four best individuals in the last generation, in addition to one in the middle; individual 16, as well as the last one; number 32. The last two were chosen to prevent only testing out solutions close to each other in gene values, as this can be the case when using GA optimisation for a sizeable number of generations.

5.10.1 Analysis

As has been discussed in 5.9, the gait parameters found using this approach are close to the default parameters, especially for the maximum step length in x- and y-directions. The results from this trial therefore have quite similar performances as when walking using the default gait parameters. When looking at the four best individuals, they walk similar lengths in the forwards direction, where number one has a value of 76.4 centimetres

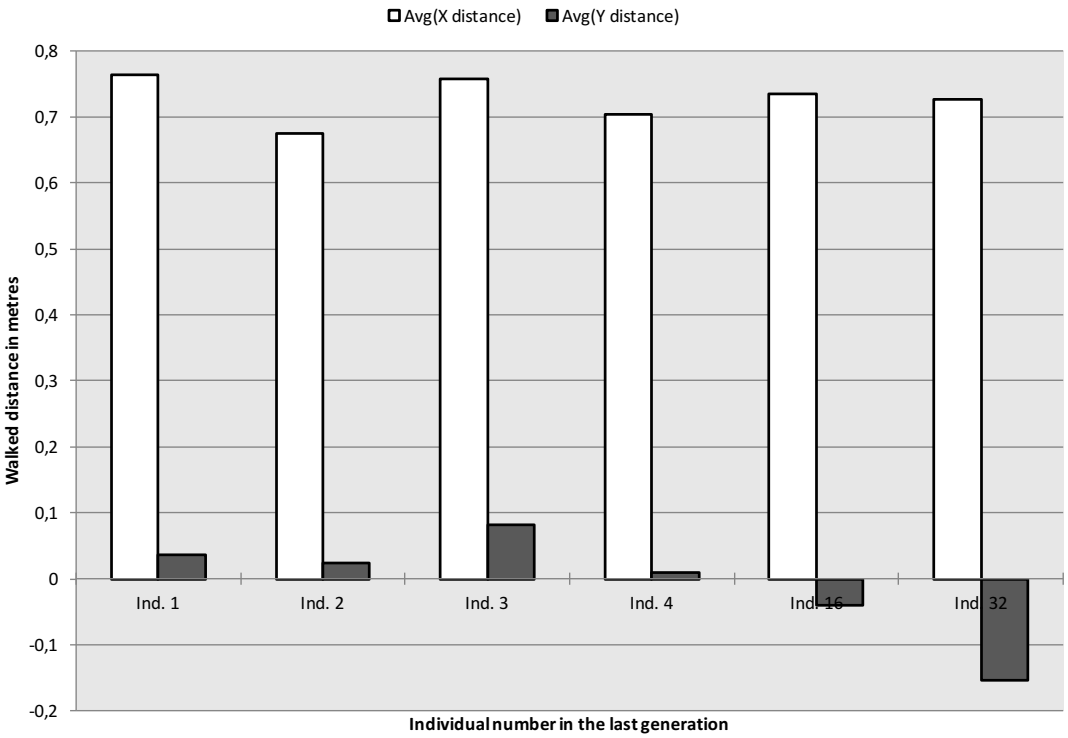


Figure 5.12: Results on the flat carpet floor in the MoCap lab using gaits optimised on uneven terrain for some selected solutions in the last generation.

in average over the five conducted trials. This gives a speed of 0.0764 centimetres per second. However, all of them walks slower than the default parameters, probably due to the terrain being used for optimisation, where using the default parameters made the robot fall over. The performance of walking on the carpet floor using the default parameters can be found in Figure 5.8. The sideways movement is low, with solution number four having a remarkable low value of only 0.88 centimetres over the 10 second evaluation time on average during the 5 trials. This is as expected in Figure 5.3, suggesting low fitness values to be highly transferable, and further backs up the impression gained throughout the project.

The two last solutions have similar performances to the first four in forwards movement, but have a negative sideways movement, meaning they walk out to the right instead of to the left. This is the first time there have been observed walks going in this direction, although the experiment found in section 5.8 showed a right sideways movement in simulation for some parameters. The reason for this behaviour is unclear, and by looking at the actual optimised parameters for these two solutions, found in Table 5.6, one can see that the maximum step lengths and rotation parameters are fairly similar to the default ones, in addition to the step height. The

maximum step frequency parameters should not have any effect on this sideways walking behaviour either, and that leaves one with only the maximum torso rotation behaviours in x- and y-direction. Unfortunately, these are quite different from each other, giving no clue on this behaviour.

MaxStepX	**MaxStepY**	**MaxStepTheta**	**MaxStepFreq**	**StepHeight**	**TorsoWX**	**TorsoWy**
0.034	0.146	0.441	0.906	0.014	0.050	0.063
0.036	0.157	0.021	0.757	0.015	0.019	0.056
0.040	0.140	0.349	1.000	0.020	0.000	0.000

Table 5.6: Top and middle rows: The parameters used to obtain the results for solution 16 and 32, respectively, found in Figure 5.12. The bottom row lists the default parameters.

Chapter 6

Discussion

This chapter contains a more general discussion about the results presented in the previous chapter, as well as conclusion part, before describing suggestions for future work.

6.1 General discussion

By looking at the results presented, it seems like the conducted approaches are able to find good results for bipedal locomotion.

6.1.1 Preliminary investigation

As for the preliminary investigation conducted, the first implemented simple GA did not achieve satisfying results. The reasons for the seemingly non-deterministic results can be many, but the mentioned problems with syncing the evaluation time between the control system and the simulator, as well as not properly resetting the robot between the trials can have impacted the result in a negative way. Since the implementation used an elitism rate of 0.1 by randomly selecting 10 percent of the population for survival into the next generation, promising solutions can have been discarded, such as the one with highest obtained fitness found after about 190 generations. The NSGA-II was therefore decided to be used for the remaining experiments, including the ones that only had one objective to be optimised. The NSGA-II incorporates an elitism rate of 1, meaning that potentially all the solutions in the current generation can survive, if no improvement is found. When transferring the solutions onto the real robot, it was discovered that the underlying surface did not affect the results in a major degree. Although there was observed differences between the gaits when running the robot on laminated and carpet floor, the mentioned measurement techniques probably plays the major role of the different performances. Another observation was that the fastest gaits had a problem with walking further and further out to the left during the walk,

suggesting a potential Reality Gap between the simulation and reality. This was further backed up by later experiments, and the differences in fitness values seems to grow with higher fitness values, suggesting that the fastest evolved gaits exploits features not modelled well in the simulator.

6.1.2 Transferability Approach

The results for dealing with the mentioned Reality Gap problem using the Transferability Approach showed the ability of reducing the gap in some extent, particularly in the walked forwards distance. It also seems like the difference in sideways movement was reduced a bit, although the specific transferability function used probably can have impacted the performance of using this approach. Due to the discovery that the inputs and outputs of the function in practice uses the same data, it probably will be a good idea to make another function to further test out this approach. When ignoring the problem with the robot walking sideways, the fastest gaits using this and the previous approach achieved speeds of around 0.13 cm/s, a bit faster than what have been managed before, as mentioned in the papers in the Background chapter.

6.1.3 Optimisation on uneven terrain

When it comes to the bipedal locomotion optimisation done on a more uneven terrain, the obtained gait parameters were found to be quite similar to the default values. The reasons for this have already been discussed, but using a different and possibly dynamic terrain can make this approach more powerful. Due to the more complex optimisation problem, running the evaluation for a longer time was conducted, as the increase in fitness values took a longer time. The resulting gaits was afterwards transferred onto the real robot, but only tested on a flat surface. This resulted in a very stable gait, and some of the transferred individuals showed an impressively straight walk, but with the weakness of being slower than the default gait. Testing out the gaits on a similar terrain to that being used for optimisation should also be of interest, to see if the optimisation actually managed to make the robot proceed over such a terrain, also in real life. This approach also managed to optimise some gaits having a negative sideways walk, i.e. walking out to the right, which differentiates them from the other gaits optimised in the other experiments. The reason for this behaviour was not found, as the parameters gave no clue of why this happened.

6.2 Conclusion

The stated main purpose of this project has been to investigate the Nao robot's suitability ford doing Evolutionary Robotics experiments. As a

part of this, it was of interest to implement an evolutionary learning platform capable of doing such research, in addition to finding interesting experiments. During the work, an implementation of such a learning platform has been developed, with the ability of testing out different Evolutionary Algorithms and experiments. It is module-based, making it easy to implement new functionality in the future. As for the experiments, overall they can be said to be a success, with the first part of the locomotion optimisation being able to find a faster gait than what is achieved by default in simulation. Other aspects of the implementation, such as the ability to conduct experiments both in simulation and reality is also worth mentioning. The simulator used in the thesis has many interesting features, such as the ability to model the real Nao robot and at the same time making it accesible using the official API, although the robot supports using other simulators as well.

The test phase when transferring these gaits onto the real robot, however revealed a significant Reality Gap. Methods to deal with this was therefore explored, such as the Transferability Approach. Experiments revealed it's capability of reducing this gap, although the function used probably can be questionable. Nevertheless, the approach seems promising for further experiments, and conducting experiments using other transferability functions can be of interest in the future.

When it comes to the data collection part, the described simulator and Motion Capture equipment's capabilities have been found suitable for the conducted experiments. Their abilities to gather accurate information such as omni-directional walking distances have been found useful, when comparing the gaits found using different EAs and approaches, as well between simulation and reality.

As a final word, the Nao robot and it's platform can be said to be suitable for doing Evolutionary Robotics research. The presented research papers, along with the conducted experiments have revealed successful results in the work of optimising good and fast walking gaits, something which is an important challenge in ER. The implemented evolutionary learning platform has made it easy to continue the investigation of the platform, with the abilities of easily adding new features and Evolutionary Algorithms for future research.

6.3 Future work

This section is about future interesting experiments and work that has not been been conducted in this thesis, but would have been if there was more time left.

There are several interesting ideas and suggestions which have appeared during the work on the thesis, and some of them would as a consequence have been conducted if there had been more time left. The other points

are more interesting in the context of doing future investigations and experiments on the Nao platform, i.e. being a natural continuation of this described project. All of the points are listed below:

- More runs for the evolutionary optimisation processes.
- Optimise in simulator faster than real-time.
- Alter the joints directly using a more low-level approach.
- Optimise in simulator using several robots in parallel.
- Test out other walking methods for locomotion optimisation.
- Explore and conduct other kinds of motion experiments.
- Use another Nao robot to investigate the sideways movement problem, as well as making another transferability function.
- Optimise locomotion gaits on other uneven terrains.
- Locomotion gait optimisation focusing on other factors such as energy consumption.

Maybe the most important point here is the first one, saying that it would be desirable to run the Evolutionary Optimisations several times, and not only once. This will be necessary in order to make the results from them more robust. As the results of EAs can be different between runs due to the randomness of their operators, this is especially important when one wants to compare the performance of several Evolutionary Algorithms with each other, making the differences statistically significant, removing any impacts from this randomness. The next two points in the list is a consequence of the first one, as having the need of doing many runs per optimisation problem can take a significant amount of time. The possibilities of simulating faster than real-time in the Webots simulator and others can be found in chapter 3.2.4 on page 42, using a more low-level approach by altering the joint positions directly. To simulate several robots in parallel should be possible with some small modifications to the learning platform, either by placing several distinct robots in the same simulation, or to run a number of simulator windows at once.

Other interesting experiments involves testing out different walking methods than the ones presented for the locomotion gait optimisation, as well as other kind of motion experiments. Examples of other experiments can be to explore the robot's possibilities and optimise e.g.crawling or jumping. The next point in the list is about the physical Nao robot that has been utilised in the experiments. As previously mentioned, it has a problem with walking more and more to the left, even with the default gait parameters. Doing the same experiments on another Nao will show if this is a general problem or behaviour for the Nao on the tested surfaces, or if it is specific to the Nao used in this project. Another transferability function can also be made to investigate this problem further. Another point is about the locomotion gait optimisations done on an uneven terrain.

Using the suggested terrains such as placing small obstacles and cables on a flat surface for optimisation, or even use a dynamically changing terrain or with a random starting position can also be of interest when focusing on the gait stability. The last point is about using other fitness functions when evaluating the gaits, such as e.g. focus on energy consumption or the smoothness of the walk.

Bibliography

[1] Kazuhiko Akachi et al. 'Development of humanoid robot HRP-3P'. In: *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*. IEEE. 2005, pp. 50–55.

[2] Aldebaran. *NAO robot: intelligent and friendly companion | Aldebaran*. Aug. 2014. URL: http://www.aldebaran.com/en/humanoid-robot/nao-robot.

[3] Aldebaran. *Romeo robot project | Aldebaran*. Feb. 2015. URL: https://www.aldebaran.com/en/robotics-company/projects.

[4] Seshadri Aravind. 'A fast elitist multiobjective genetic algorithm: NSGA-II'. In: *IEEE Transactions on Evolutionary Computation* (2004).

[5] Josh C Bongard. 'Evolutionary robotics'. In: *Communications of the ACM* 56.8 (2013), pp. 74–83.

[6] Josh C Bongard and Hod Lipson. 'Nonlinear system identification using coevolution of models and tests'. In: *Evolutionary Computation, IEEE Transactions on* 9.4 (2005), pp. 361–384.

[7] Bruno Siciliano. *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008.

[8] Birgitta Burger and Petri Toiviainen. 'MoCap Toolbox – A Matlab toolbox for computational analysis of movement data'. In: *Proceedings of the 10th Sound and Music Computing Conference*. Stockholm, Sweden: KTH Royal Institute of Technology, 2013, pp. 172–178.

[9] TIOBE Software BV. *TIOBE Index for February 2015*. Feb. 2015. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[10] Angelo Cangelosi, Matthew Schlesinger and Linda B Smith. *Developmental robotics: From babies to robots*. 2015.

[11] Carlos A Coello Coello. 'Evolutionary multi-objective optimization: a historical view of the field'. In: *Computational Intelligence Magazine, IEEE* 1.1 (2006), pp. 28–36.

[12] Steve Collins et al. 'Efficient bipedal robots based on passive-dynamic walkers'. In: *Science* 307.5712 (2005), pp. 1082–1085.

[13] Technical Committee. *RoboCup Standard Platform League (NAO) Rule Book*. Aug. 2014. URL: http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2014.pdf.

[14] Stefano Nolfi Dario Floreano Phil Husbands. 'Evolutionary Robotics'. In: Bruno Siciliano. *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008, p. 1428.

[15] Stefano Nolfi Dario Floreano Phil Husbands. 'Evolutionary Robotics'. In: Bruno Siciliano. *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008, p. 1308.

[16] Hugo De Garis et al. 'A world survey of artificial brain projects, Part I: Large-scale brain simulations'. In: *Neurocomputing* 74.1 (2010), pp. 3–29.

[17] Kalyanmoy Deb and Ram Bhushan Agrawal. 'Simulated binary crossover for continuous search space'. In: *Complex systems* 9.2 (1995), pp. 115–148.

[18] Kalyanmoy Deb and Himanshu Jain. 'An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part i: Solving problems with box constraints'. In: *Evolutionary Computation, IEEE Transactions on* 18.4 (2014), pp. 577–601.

[19] Stéphane Doncieux, Nicolas Bredèche and Jean-Baptiste Mouret. *New Horizons in Evolutionary Robotics: Extended Contributions from the 2009 EvoDeRob Workshop*. Vol. 341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, p. 3, 4, 5, 7 and others. ISBN: 9783642182716.

[20] Boston Dynamics. *Atlas - The Agile Anthropomorphic Robot*. Feb. 2015. URL: http://www.bostondynamics.com/robot_Atlas.html.

[21] Malachy Eaton. *Evolutionary Humanoid Robotics*. SpringerBriefs in Intelligent Systems. Springer, 2015. ISBN: 978-3-662-44598-3.

[22] The Editors of Encyclopædia Britannica. *Natural selection*. Feb. 2015. URL: http://www.britannica.com/EBchecked/topic/406351/natural-selection.

[23] Engagdet. *Nao robot to become even more of a chatterbox with new software*. Aug. 2014. URL: http://www.engadget.com/2013/10/30/nao-robot-new-nuance-voice-software.

[24] The Robocup Federation. *RoboCup*. Feb. 2015. URL: http://www.robocup.org.

[25] Dario Floreano and Francesco Mondada. 'Evolutionary neurocontrollers for autonomous mobile robots'. In: *Neural networks* 11.7 (1998), pp. 1461–1478.

[26] Open Source Robotics Foundation. *Nao - ROS Wiki*. Mar. 2015. URL: http://wiki.ros.org/nao.

[27] Open Source Robotics Foundation. *ROS.org - About ROS*. Mar. 2015. URL: http://www.ros.org/about-ros.

[28] Daoxiong Gong, Jie Yan and Guoyu Zuo. 'A review of gait optimization based on evolutionary computation'. In: *Applied Computational Intelligence and Soft Computing* 2010 (2010).

[29] Evert Haasdijk et al. 'Evolutionary robotics'. In: *Evolutionary Intelligence* 7.2 (2014), pp. 69–70.

[30] K. H. Han and J. H. Kim. 'Quantum-inspired evolutionary algorithm for a class of combinatorial optimization'. In: *IEEE Transactions on Evolutionary Computation* 6.6 (2002).

[31] K. H. Han and J. H. Kim. 'Quantum-inspired evolutionary algorithms with a new termination criterion, H-epsilon gate, and two-phase scheme'. In: *IEEE Transactions on Evolutionary Computation* 8.2 (2004).

[32] American Honda Motor Co. Inc. *ASIMO Specifications*. Feb. 2015. URL: http://asimo.honda.com/asimo-specs/.

[33] Kawada Industries Inc. *Humanoid Robot HRP-4*. Feb. 2015. URL: http://global.kawada.jp/mechatronics/hrp4.html.

[34] ROBOTIS Inc. *ROBOTIS DARwIn-OP*. Feb. 2015. URL: http://www.robotis.com/xe/darwin_en.

[35] Jacky Baltes. 'Omnidirectional walking using zmp and preview control for the nao humanoid robot.' In: *RoboCup 2009: Robot Soccer World Cup XIII*. Vol. 5949. Springer-Verlag Berlin Heidelberg, 2010, pp. 378–389. ISBN: 9783642118760.

[36] N. Jakobi. *Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics, Proceedings of the 4th European Conference on Artificial Life, ed. by P. Husbands, I. Harvey*. MIT Press, Cambridge 1997, pp. 348–357.

[37] N. Jakobi, P. Husbands and I. Harvey. 'Noise and the reality gap: The use of simulation in evolutionary robotics'. In: *Advances in Artificial Life* 929 (1995), pp. 704–720.

[38] Satoshi Kagami et al. 'A fast generation method of a dynamically stable humanoid robot trajectory with enhanced zmp constraint'. In: *Proc. of IEEE International Conference on Humanoid Robotics (Humanoid2000)*. 2000.

[39] Shuuji Kajita and Kazuo Tani. 'Experimental study of biped dynamic walking in the linear inverted pendulum mode'. In: *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. Vol. 3. IEEE. 1995, pp. 2885–2891.

[40] Shuuji Kajita et al. 'A realtime pattern generator for biped walking'. In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 1. IEEE. 2002, pp. 31–37.

[41] Shuuji Kajita et al. 'Biped walking pattern generation by using preview control of zero-moment point'. In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 2. IEEE. 2003, pp. 1620–1626.

[42] Sameer Agarwal Kalyanmoy Deb Amrit Pratap and T. Meyarivan. 'A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II'. In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197.

[43] Nate Kohl and Peter Stone. 'Policy gradient reinforcement learning for fast quadrupedal locomotion'. In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Vol. 3. IEEE. 2004, pp. 2619–2624.

[44] Sylvain Koos, J-B Mouret and Stéphane Doncieux. 'The transferability approach: Crossing the reality gap in evolutionary robotics'. In: *Evolutionary Computation, IEEE Transactions on* 17.1 (2013), pp. 122–145.

[45] J. Kulk and J. Welsh. 'A low power walk for the nao robot'. In: *Australasian Conference on Robotics and Automation*. 2008.

[46] J. Kulk and J. Welsh. 'Autonomous optimisation of joint stiffnesses over the entire gait cycle for the nao robot'. In: *International Symposium on Robotics and Intelligent Sensors*. 2010.

[47] Jason Kulk and James S Welsh. 'Evaluation of walk optimisation techniques for the nao robot'. In: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE, pp. 306–311.

[48] Siavash Haroun Mahdavi and Peter J Bentley. 'An evolutionary approach to damage recovery of robot motion with muscles'. In: *Advances in Artificial Life*. Springer, 2003, pp. 248–255.

[49] Stephen Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2014.

[50] Vitor Matos and Cristina P Santos. 'Towards goal-directed biped locomotion: Combining CPGs and motion primitives'. In: *Robotics and Autonomous Systems* 62.12 (2014), pp. 1669–1690.

[51] J. Morimoto et al. 'A biologically inspired biped locomotion strategy for humanoid robots: Modulation of sinusoidal patterns by a coupled oscillator model'. In: *IEEE Transactions on Robotics* 24.1 (2008).

[52] Jean-Baptiste Mouret, Sylvain Koos and Stéphane Doncieux. 'Crossing the reality gap: a short introduction to the transferability approach'. In: *arXiv preprint arXiv:1307.1870* (2013).

[53] Stefano Nolfi. 'Genetic algorithms'. In: Stefano Nolfi. *Evolutionary robotics: the biology, intelligence, and technology of selv-organizing machines*. MIT Press, 2004, pp. 19–27. ISBN: 0262640562.

[54] Yu Ogura et al. 'Development of a new humanoid robot WABIAN-2'. In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE. 2006, pp. 76–81.

[55] Chang-Soo Park, Young-Dae Hong and Jong-Hwan Kim. 'An evolutionary central pattern generator for stable bipedal walking by the increased double support time'. In: *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. IEEE. 2011, pp. 497–502.

[56] Jordan B Pollack et al. 'Evolutionary techniques in physical robotics'. In: *Evolvable Systems: from biology to hardware*. Springer, 2000, pp. 175–186.

[57] Aldebaran Robotics. *NAO Data Sheet*. Sept. 2014. URL: http://www.aldebaran.com/sites/aldebaran/files/nao_datasheet.pdf.

[58] Aldebaran Robotics. *Overview - Aldebaran 2.1.2.17 documentation*. Mar. 2015. URL: http://doc.aldebaran.com/2-1/index.html.

[59] UCLA Robotics and Mechanisms Laboratory. *DARwIn OP: Open Platform Humanoid Robot for Research and Education*. Feb. 2015. URL: http://www.romela.org/main/DARwIn_OP:_Open_Platform_Humanoid_Robot_for_Research_and_Education.

[60] RoMeLa. *RoMeLa RoboCup 2012*. Feb. 2015. URL: http://romelarobocup.blogspot.mx/2012/06/team-darwin-repeats-win-at-robocup-in.html.

[61] SJ Russell, P Norvig and E Davis. 'Artificial intelligence: a modern approach'. In: *Prentice Hall series in artificial intelligence* (2010).

[62] Yoshiaki Sakagami et al. 'The intelligent ASIMO: System overview and integration'. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2002, pp. 2478–2483.

[63] J David Schaffer. 'Multiple objective optimization with vector evaluated genetic algorithms.' In: *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*. 1985, pp. 93–100.

[64] Nima Shafii, Nuno Lau and Luis Paulo Reis. 'Learning to Walk Fast: Optimized Hip Height Movement for Simulated and Real Humanoid Robots'. In: *Journal of Intelligent & Robotic Systems* (2015), pp. 1–17.

[65] Donald Shepard. 'A two-dimensional interpolation function for irregularly-spaced data'. In: *Proceedings of the 1968 23rd ACM national conference*. ACM. 1968, pp. 517–524.

[66] Manuel Fernando Silva and JA Tenreiro Machado. 'A literature review on the optimization of legged robots'. In: *Journal of Vibration and Control* 18.12 (2012), pp. 1753–1767.

[67] Pedro Silva et al. 'Automatic generation of biped locomotion controllers using genetic programming'. In: *Robotics and Autonomous Systems* 62.10 (2014), pp. 1531–1548.

[68] Russel Smith. *Open Dynamics Engine - home*. Mar. 2015. URL: http://www.ode.org.

[69] Nidamarthi Srinivas and Kalyanmoy Deb. 'Muiltiobjective optimization using nondominated sorting in genetic algorithms'. In: *Evolutionary computation* 2.3 (1994), pp. 221–248.

[70] Stefano Nolfi. *Evolutionary robotics: the biology, intelligence, and technology of selv-organizing machines*. MIT Press, 2004. ISBN: 0262640562.

[71] Bjarne Stroustrup. *Bjarne Stroustrup's FAQ*. Mar. 2015. URL: http://www.stroustrup.com/bs_faq.html.

[72] Tomomichi Sugihara, Yoshihiko Nakamura and Hirochika Inoue. 'Real-time humanoid motion generation through ZMP manipulation based on inverted pendulum control'. In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 2. IEEE. 2002, pp. 1404–1409.

[73] Thomas A. McMahon. *Muscles, reflexes, and locomotion*. Princeton University Press, 1984. ISBN: 0691083223.

[74] Adrian Thompson. *Artificial evolution in the physical world*. 1997.

[75] Ernesto Torres and Leonardo Garrido. 'Automated Generation of CPG-Based Locomotion for Robot Nao'. In: *RoboCup 2011: Robot Soccer World Cup XV*. Springer Berlin Heidelberg, 2012, pp. III, XI, 461–471.

[76] Verne T. Inman. *Human walking*. Williams & Wilkins, 1981. ISBN: 068304348X.

[77] Miomir Vukobratović and Branislav Borovac. 'Zero-moment point—thirty five years of its life'. In: *International Journal of Humanoid Robotics* 1.01 (2004), pp. 157–173.

[78] Webots. *http://www.cyberbotics.com*. Ed. by Cyberbotics Ltd. Commercial Mobile Robot Simulation Software. URL: http://www.cyberbotics.com.

[79] P-B Wieber. 'Trajectory free linear model predictive control for stable walking in the presence of strong perturbations'. In: *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*. IEEE. 2006, pp. 137–142.

[80] Yuan Xu and Hedayat Vatankhah. 'SimSpark: An Open Source Robot Simulator Developed by the RoboCup Community'. In: *RoboCup 2013: Robot World Cup XVII*. Springer, 2014, pp. 632–639.

[81] J. K. Yoo, B. J. Lee and J. H. Kim. 'Recent progress and development of the humanoid robot HanSaRam'. In: *Robotics and Autonomous Systems* 57.10 (2009).

[82] Milos Zefran and Vijay Kumar. 'Two methods for interpolating rigid body motions'. In: *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*. Vol. 4. IEEE. 1998, pp. 2922–2927.